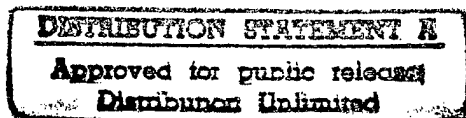


# A Reinforcement Learning Approach to Control

STTR Phase I Final report

Contract No.: N00014-96-C-0321

May 31, 1997



Prepared for:

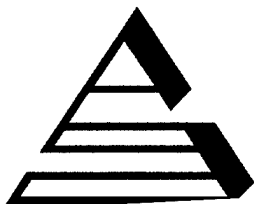
Office of Naval Research

Prepared by:

Amherst Systems Inc. and The State University of New York at Buffalo

Office of  
Naval Research

800 N. Quincy St., Arlington, VA 22217-5660



19970609 018

DTIC QUALITY INSPECTED

DTIC QUALITY INSPECTED 3





AMHERST SYSTEMS INC.

---

# A REINFORCEMENTR LEARNING APPROACH TO CONTROL

## FINAL REPORT

---


May 31, 1997

Document Control Number 62-9160004, Rev A

Prepared for: Office of Naval Research  
800 North Quincy Street  
Arlington, Virginia 22217-5660

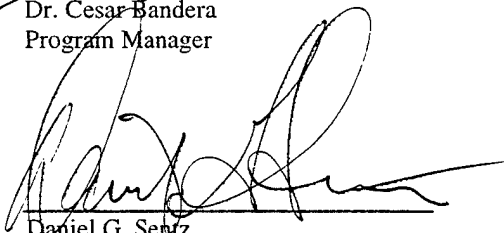
Contract No.: N00014-96-C-0321

CDRL Item: 0002



---

Dr. Cesar Bandera  
Program Manager



---

Daniel G. Senz  
Quality Assurance Manager

## Table of Contents

<b>ABSTRACT .....</b>	<b>2</b>
<b>SECTION 1: INTRODUCTION .....</b>	<b>3</b>
<b>SECTION 2: REINFORCEMENT LEARNING AND GENERAL FUNCTION APPROXIMATORS .....</b>	<b>6</b>
2.1 REINFORCEMENT LEARNING .....	6
2.2 ASSOCIATIVE VERSUS NON-ASSOCIATIVE LEARNING .....	6
2.3 IMMEDIATE VERSUS DELAYED REINFORCEMENT LEARNING .....	7
2.4 EXPLOITATION VERSUS EXPLORATION .....	8
2.5 DIRECT VERSUS INDIRECT LEARNING .....	8
2.6 $Q(\lambda)$ LEARNING .....	9
2.7 FUNCTION APPROXIMATORS .....	9
<b>SECTION 3: FOUNDATIONS OF REINFORCEMENT LEARNING.....</b>	<b>11</b>
3.1 STOCHASTIC DYNAMIC PROGRAMMING .....	11
3.2 SYNCHRONOUS VALUE ITERATION .....	13
3.3 POLICY IMPROVEMENT .....	14
3.4 ASYNCHRONOUS VALUE ITERATION AND Q-LEARNING .....	15
3.5 DISCUSSION .....	16
<b>SECTION 4: PARTIAL STATES AND LEARNED FEATURES.....</b>	<b>17</b>
4.1 ACTIVE PERCEPTION AS A PARTIALLY OBSERVED MARKOV DECISION PROCESS (POMDP) .....	17
4.2 HISTORY AUGMENTATION OF INCOMPLETE STATE FEEDBACK FOR POMDP'S .....	19
4.3 A TOKEN GAZE CONTROL PROBLEM .....	20
4.4 SIMULATION RESULTS .....	22
4.5 DISCUSSION .....	23
<b>SECTION 5: SCALING UP - PARTIAL STATES AND FIXED FEATURES.....</b>	<b>25</b>
5.1 PROBLEM DESCRIPTION .....	26
5.2 ACTIONS .....	29
5.3 REFERENCE POLICIES .....	29
5.4 FIXED FEATURE ALGORITHMS.....	31
5.5 RESULTS .....	35
5.6 DISCUSSION .....	42
<b>SECTION 6: ERROR RESIDUAL VERSUS SARSA LEARNING.....</b>	<b>45</b>
6.1 REINFORCEMENT LEARNING WITH FUNCTION APPROXIMATION.....	45
6.2 RESIDUAL ALGORITHMS AND SARSA .....	46
6.3 THE RANDOM WALK PROBLEM.....	47
6.4 THE 2-D STOCHASTIC CONTINUOUS GRID-WORLD PROBLEM .....	49
6.5 DISCUSSION .....	54
<b>SECTION 7: CONCLUSIONS.....</b>	<b>55</b>
<b>SECTION 8: REFERENCES .....</b>	<b>56</b>

## Abstract

Active perception strategies are necessary for goal-driven allocation of available resources to improve relevant information acquisition and optimize overall system performance. In addition to being both goal and data driven, these strategies must also account for the fact that information acquisition is inherently a partially observable Markov decision problem. This report describes an efficient, scalable reinforcement learning approach to the control of autonomous active vision which also satisfies the more stringent requirements of foveal machine vision. Foveal vision offers images with both wide field-of-view, useful for rapid detection, and a high acuity zone, useful for accurate recognition, without the overhead and errors inherent in dynamic registration of data from multiple sensors. However, space-variant data acquisition inherent with foveal retinotopologies necessitates deployment of refined intelligent gaze control techniques. This report first lays a theoretical foundation for reinforcement learning. It then introduces the SARSA algorithm in conjunction with history augmentation as an effective learning control method for visual attention. A general and flexible algorithm utilizing recurrent neural nets to learn relevant history features is developed and demonstrated in the context of a small (token) gaze control simulation. For high-dimensional world states, the simultaneous learning of history features and reinforcement signals may require unacceptably long training exposures before the learned behaviors become effective. A modification of the basic algorithm in which performance-adaptive feature learning is replaced by fixed features is next shown to define a simplified reinforcement learning scheme which performs effectively in simulated gaze control problems scaled to practical dimensions. The system is shown to perform well in both high and low SNR ATR environments. Since commercially feasible gaze control requires extensive generalization power, the report concludes with a description of several numerical experiments designed to evaluate the relative efficiency of various reinforcement learning algorithms and techniques for input generalization, using both prediction and control problems. Reinforcement learning coupled with history features appears to be both a sound foundation and a practical scalable base for gaze control.

## Section 1: Introduction

Control systems for autonomous or semi-autonomous platforms must maintain closed-loop system integrity and performance over a wide range of operating conditions. This objective can be difficult to achieve due to the complexity of both the plant and the performance objectives, the presence of uncertainty, and most of all limited *a priori* model information. In the design of control strategies for *active perception*, that is, the performance-adaptive choice of future observations in light of current goals, needs and knowledge state, typically all of these complications must be considered. Under such conditions, it is very difficult or even impossible to design a fixed control policy that meets the desired performance specifications. From a control-theoretic standpoint, these difficulties may result from nonlinear or time-varying behavior, poorly modeled environment and plant dynamics, high dimensionality, multiple inputs and outputs, complex objective functions, constraints, and the possibility of actuator or sensor failures. Each of these effects, if present, must be addressed if the system is to operate properly in an autonomous or semi-autonomous fashion for extended periods of time.

The system designer may take several actions: (1) reduce the desired level of performance, (2) conduct additional theoretical or empirical model development in advance to reduce uncertainty, or (3) design the control system to adjust itself on-line automatically to reduce uncertainty and/or improve performance. The third action is significantly different from the first two because the resulting design is not fixed, and instead has inherent operational flexibility.

This Phase I effort is concerned with techniques that are intended for those difficult situations, typical of active perception and autonomous behaviour, where it is unacceptable on the basis of requirements, cost, or feasibility, to reduce the desired level of closed-loop system performance. As a consequence, the system designer can only increase the level of achievable performance by reducing uncertainty, and uncertainty can only be reduced on-line, through direct interaction with the actual system. The novel line of investigation under this Phase I effort is to create an autonomous robot that learns to act intelligently in unstructured environments or environments with uncertainty, much like humans do.

Instead of designing a complete and fixed<sup>1</sup> control system from insufficient *a priori* information, control laws for poorly modeled nonlinear time-varying systems are learned through direct interaction between the system and its environment, and effective use of this operational experience. As a result, learning control offers several unique features that make it attractive for building systems, such as active perception controllers, that must operate in uncertain environments.

It is important to realize that many learning tasks arising in control require methods that cannot be precisely characterized as *supervised learning*. Imagine that one wishes to adjust a control law in order to improve the performance of a plant as measured by a performance criterion that evaluates the overall behavior of the plant; for example, one might wish to minimize a measure of the plant's

---

1. The term "fixed" is used here to indicate those control systems for which the parameters and structure are determined in an open-loop (performance independent) fashion.

energy cost over time. Optimal control methods apply if there models of the plant and performance measure that are sufficiently complete, accurate and tractable. However, in less structured situations it is possible to improve plant performance over time by means of on-line methods, performing what is called *reinforcement learning*. Whereas the performance measure for supervised learning is defined in terms of a set of desired targets by means of a known error criterion such as mean-squared error, reinforcement learning is concerned with the problem of learning from rewards and punishments (see below for a detailed discussion). Therefore, in tasks of this kind there exist desired control signals – those that lead to optimal plant performance – but the learning system is not told what they are because there is no teacher available. The problem is to find these optimal control signals, not simply to remember and generalize from them. It is this kind of learning control problems that this Phase I effort addresses.

Note that both adaptive and learning control systems are capable of automatically adjusting their internal representations on-line, a process achieved either directly by manipulating the control law itself, or indirectly through model identification and control law redesign. Also, both make use of performance feedback information gained through closed-loop interactions with the plant and its environment. However, the major difference is the range of applicability. A adaptive control system, one with fixed structure and variable parameters, can respond effectively to a limited range of disturbances and environmental changes (those consistent with its fixed structure). More general perturbations cannot be adapted by parametric degrees of freedom only. Greater robustness requires non-parametric, structural variability, the hallmark of learning systems. Due to its interest in autonomous behavior and active perception, this Phase I effort addresses learning control rather than adaptive control strategies.

The technique we introduce here is a combination of reinforcement learning and biologically inspired function approximators such as neural networks, both of which have been used successfully in a wide variety of tasks ranging from predictions to nonlinear dynamic modeling. In particular, reinforcement learning by itself can produce useful behaviors without higher level command (e.g., semantics). More specifically, we describe efficient learning control algorithms that combine  $Q(\lambda)$ -like learning [PengWilliams94,96,Bandera96] or State-Action-Reward-State-Action (SARSA) [RummeryNiranjan94] techniques with sparse-coded function approximators such as the Cerebellar Model Articulation Controller (CMAC) [Albus75, MillerGlanzKraft87] and Radial Basis Function neural net (RBF) [Powell87]. While reinforcement learning addresses the problem of improving performance as evaluated by *any* measure whose values can be supplied to the learning system, general approximators such as CMACs are compact representation schemes that are capable of implementing any function to any desired degree of accuracy given sufficient resources. The key benefits of the proposed technique are its ability to improve plant performance over time in less structured environments, its robustness and learning efficiency, and its scalability to large-scale nonlinear problems. In addition, it is feasible to construct fast, parallel devices to implement this technique for real-time applications. Such abilities are the prerequisite for any system to operate in real world conditions. Furthermore, such algorithms, when combined with a plant model, may significantly reduce the overall cost of the development of gaze control strategies.

The body of the report is organized as follows: Section 2 gives a brief introduction to machine learning in general and reinforcement learning in particular, including issues associated with reinforcement learning. Section 3 introduces a mathematical framework, namely *dynamic program-*

*ming*, from which various reinforcement learning techniques are derived. Section 4 describes an experiment designed to evaluate reinforcement learning using embedded recurrent neural networks tasked to learn to extract relevant history features using online rewards and punishments. Motivated by scaling issues, in Section 5 reinforcement learning experiments with fixed rather than learned history features are described. Section 6 compares two reinforcement learning schemes incorporating generalization: SARSA and residual learning. Finally, overall conclusions and references, Sections 7 and 8 respectively, complete the report.

## Section 2: Reinforcement Learning and General Function Approximators

This section provides a brief overview of reinforcement learning and general function approximation schemes. A more complete survey of the subject is presented in [Girosi95,Kaelbling96].

### 2.1 Reinforcement Learning

Learning is defined in this proposal as the capability of any system to change itself over time with the intent of improving performance on tasks defined by its environment. Most learning problems can be divided into three broad categories: *Unsupervised*, *Supervised*, and *Reinforcement* learning problems. Unsupervised learning is concerned with clustering input data into categories that can later be used to predict future data. It receives no feedback from outside and is the environment that provides an implicit feedback. In contrast, supervised learning uses feedback provided by an external teacher to learn input-output mappings. In supervised learning, the system receives an input instance, produces an output, receives the desired output from the teacher, and then uses the discrepancy between its actual output and the desired output to make an adjustment to itself. Reinforcement learning is similar to supervised learning in that it receives an outside feedback. However, the nature of the feedback is different. The feedback is instructive in the case of supervised learning and evaluative in the case of reinforcement learning. That is, for supervised learning the system is presented with the correct output for each input instance, while for reinforcement learning the system produces a response that is then evaluated using a scalar indicating the appropriateness of the response. As an example, a checker playing program that uses the outcome of a game to improve its performance is a reinforcement learning system. Knowledge about an outcome is useful for evaluating the total system's performance, but it says nothing about which actions were instrumental for the ultimate win or loss. Learning in pattern classification is not reinforcement learning since during the training, such systems receive information from a teacher about the correct classification. In general, reinforcement learning is more widely applicable than supervised learning since any supervised learning problem can be treated as a reinforcement learning problem. In the context of building learning control systems whose behaviors are to be developed autonomously, reinforcement learning provides an attractive framework because of its similarity to the problem faced by animals that are expected to behave intelligently in situations with great uncertainty. This type of learning is also used to model behavior learning in animals and humans in experimental psychology, and is the main focus of this Phase I effort.

### 2.2 Associative versus Non-associative Learning

One variation in reinforcement learning is the distinction between associative and non-associative learning. In the non-associative reinforcement learning paradigm, reinforcement is the only information the learning system receives from its environment. That is, the system has no input information about the environment. It does not know the state of the environment in which it is embedded. In these tasks, the object of the system is to determine the optimal action that maximizes



its return, or cumulative discounted sum of future rewards. Non-associative reinforcement learning does not concern us here because our applications require taking into account the changes of environmental states. Non-associative reinforcement learning has been mainly studied in the context of function optimization [WilliamsPeng93] and learning automata theory [NarendraThathatchar89].

Associative reinforcement learning extends non-associative reinforcement learning in a natural way. In such a learning paradigm, the control system receives input information that indicates the state of its environment as well as reinforcement. The additional information tells what state of its environment the system is in. It should be pointed out that, in general, input the system receives about its environment is usually incomplete in that the input only reveals partial information about the state of its environment due to the system's limited sensory capabilities and/or limited memory of past observations. To emphasize that observations need not be in 1-to-1 correspondence with world states, the term *sensations* is often used for this output feedback.

The objective for the system in these tasks is to learn an optimal mapping from the sensations about its environment to the actions it can execute. In the simplest case, the mapping will be a function of the current sensation. In general, however, the system might try to represent states which summarize all of the past sensations. Note that, at each state, the job of associative reinforcement learning is the same as that of non-associative reinforcement learning. Associative reinforcement learning is of great interest to us here due to its similarity to modeling the learning and decision-making tasks faced by animals and humans.

### 2.3 Immediate versus Delayed Reinforcement Learning

Another complication to reinforcement learning is the timing of reinforcement. In the simplest tasks, the system receives, after each action, reinforcement indicating the goodness of that action. Immediate reinforcement occurs commonly in non-associative learning tasks. In more complex tasks, however, reinforcement is often temporally delayed, occurring only after the execution of a sequence of actions. Delayed reinforcement learning is important because in many domains, immediate reinforcement regarding the value of a decision may not always be available. For example, in a chess game, the outcome of the game is not known until the game is over. The difficulty associated with delayed reinforcement learning is that some judicious actions now may only allow high rewards to be achieved later; each action in an action sequence may be essential to achieving a reward, even though not all of the actions are followed by immediate rewards. Conversely, in a chess game, a move may lead a particular player into a "doomed" situation from which it is impossible to prevent that player from eventually losing the game - but the loss actually occurs some time later. It would be unwise to blame the actions taken immediately before the game was lost, for these actions may have been the best that could be taken in the circumstances. The actual mistake may have been made some time earlier. The problem of assigning credit or blame to decisions when reinforcement is delayed is the well known *credit assignment problem* [Minsky61]. This Phase I effort focuses on methods for efficient learning in delayed reinforcement settings.

## 2.4 Exploitation versus Exploration

Another important aspect of reinforcement learning is that a learning system needs two capabilities if it is to act optimally in its environment with reasonable efficiency: exploitation and exploration. Trying to satisfy both goals at the same time leads to a conflict. Exploitation suggests that the scope of the search should be narrowed as experience about actions accumulates, or when exhaustive search is not feasible, or when the system has only a finite life span, so that the system will be able to concentrate on performing actions that are expected to be good in order to increase overall rewards, but exploration suggests that the scope of the search should not be narrowed irrevocably so far as to let superior actions with little experience slip through permanently.

The tradeoff most reinforcement learning systems take to resolve the conflict is to occasionally take random actions. This ad-hoc approach works well for most problems encountered empirically, but has no strong theoretical foundation. A more statistically sound exploration strategy, the *Interval Estimation* (IE) algorithm [Kaelbling90], has been proposed that uses the combination of mean and confidence to choose appropriate actions. Other approaches have also been suggested [Sutton90].

## 2.5 Direct versus Indirect Learning

There are two general approaches to learning optimal policies. One is the *indirect* or *model-based* approach, which requires learning an environmental model in the form of state-transition and reward probabilities. A search or computational technique such as dynamic programming is then applied to produce an optimal policy for the system. Most work in the adaptive control of Markov processes described in the control engineering literature has been model-based. Learning a world model is usually achieved by supervised learning that is relatively simple compared to the expensive search for an optimal policy or a value function. It is interesting to note that a learned world model need not be completely accurate for dynamic programming to produce a policy that accomplishes underlying tasks. Very often an approximate model is sufficient.

The second approach to learning how to obtain optimal policies is *direct* [BartoSuttonWatkins90] or *primitive* learning [Watkins89]. Instead of learning an environmental model, a direct method adjusts the policy on the basis of its observed consequences. The system must act in its environment, try out a variety of actions, observe their consequences, and adjust its policy to improve performance over time. While some reinforcement learning procedures are indirect (eg. real time dynamic programming Barto93) most are direct. Direct controls corresponds most closely to simple reactive behaviors.

It is possible to combine direct methods with indirect methods in a variety of ways. Sutton's Dyna architecture is a classic example [Sutton90]. Also see [MooreAtkeson92,PengWilliams93]. One typical characteristic of real world problems is their high dimensionality. It would be impractical for a computational method, such as dynamic programming, to produce a complete solution before committing the very first act. In such large-scale problems, a learning method with generalized capabilities that computes good solutions quickly in certain important areas of the state space may be sufficient. Tesauro's TD Gammon [Tesauro92] suggests that direct methods appear to be good can-

didates for accomplishing this. This, however, does not imply that the advantages of model-based learning should be overlooked. In fact, in the context of learning, the combinations of direct learning and model-based learning can offer significant performance improvements over either technique alone and make learning systems most promising.

## 2.6 $Q(\lambda)$ Learning

The  $Q(\lambda)$  learning algorithm [PengWilliams94,96] is an incremental multi-step method that extends the one-step Q-learning algorithm [Watkins89] by combining it with  $TD(\lambda)$  returns ( $0 \leq \lambda \leq 1$ ) [Sutton88] for learning from delayed rewards.  $\lambda$  is a weighting parameter that determines to what extent future rewards will contribute to the estimates of current action values. By allowing corrections to be made incrementally to the predictions of observations occurring in the past, the  $Q(\lambda)$  learning method propagates information rapidly to where it is important. The  $Q(\lambda)$  learning algorithm works significantly better than the one-step Q-learning algorithm on a number of tasks; its basis in the integration of one-step Q-learning and  $TD(\lambda)$  returns makes it possible to take advantage of some of the best features of both the Q-learning and actor-critic learning paradigms, and to be a potential bridge between them. It can also serve as a basis for developing various multiple time-scale learning mechanisms that are essential for applications of reinforcement learning to large-scale problems.

## 2.7 Function Approximators

Global function approximations assume that there is a underlying, learnable structure in the problem. This assumption is essential in order for a generalization to make sense. Some approximation methods, such as polynomial approximations and neural networks [Barnhill77, Franke82, Schumaker76], use a strong form of the assumption. Global models that are sufficient for capturing the structure of the problem being modeled are usually used to fit entire training data. The difficulty is to find an appropriate structure for a global model. One simply can not tell if a given set of parameters would be sufficient to produce an adequate model for the data, unless it is chosen with *a priori* knowledge. An additional difficulty associated with connectionist networks is that because an iterative gradient descent search procedure is used to find the appropriate set of weights, training data must be repeatedly presented to the network. In case of on-line learning, this implies that old experiences have to be stored and presented again since learning new experiences may degrade the representation of older experiences. Other methods, such as decision trees and the nearest neighbor generalization, use a weaker form of the assumption.

Sparse coarse coded function approximators, such as the CMAC and RBFs, have the same representational capability as more conventional neural networks such as multi-layer perceptrons and Hopfield nets. However, they are more efficient computationally and work well in many other respects, such as better response to spatial variations. In addition, basis functions decrease with the distance of the data points from the evaluation point, so that only the neighboring points affect the estimate of the function at the evaluation point, therefore providing a "local" approximation scheme. It should be noted however that these techniques in general do not have the highest pos-

sible degree of locality, since the parameter that controls the locality is the same for all the basis functions. It is possible to design even more local techniques, in which each basis has a parameter that controls its locality.

### Section 3: Foundations of Reinforcement Learning

Dynamic programming provides solutions to problems of credit assignment in delayed reinforcement learning and optimal control. The significance of dynamic programming has often been under-appreciated in artificial intelligence (AI) research. The appeal and large potential of dynamic programming in AI is that it provides an attractive basis for compiling actions into reactive policies that can be used for real-time control, as well as for developing methods for learning such policies when the world model is not available. It should be emphasized at the outset that although dynamic programming requires a complete model of the world, it is the framework it provides which is relevant to the synthesis and analysis of a variety of reinforcement learning algorithms. Dynamic programming has also been used in behavioral ecology for the study of animal behavior.

The fundamental principle of dynamic programming is to determine an optimal policy by constructing a *value function* or a *return function* on the state space. This chapter discusses dynamic programming in the context of Markov decision processes and introduces the major difficulties faced by dynamic programming. The discussion is based on [Peng94].

#### 3.1 Stochastic Dynamic Programming

One of the most powerful techniques developed for the solution of optimization problems, such as routing and scheduling, is dynamic programming (DP) [Bellman61]. It can be formulated as follows.

A *discrete-time Markov decision process* is a 4-tuple  $(X, A, P, R)$ , where

1.  $X$  is a finite set of states on which the process evolves.
2.  $A$  is a finite set of actions. For each state  $x \in X$  there is an associated non-empty subset of  $A$ , whose elements are the *admissible actions* when the process is in state  $x$ .
3.  $P$  is a probabilistic state-transition law. It is a function:  $X \times X \times A \rightarrow \mathfrak{R}$ , and  $P_{xy}(a)$  is the probability to reach state  $y$  from state  $x$  when action  $a$  is taken in that state. For any pair  $(x, a)$ ,  $\sum_y P_{xy}(a) = 1$ .
4.  $R: X \times A \rightarrow \mathfrak{R}$  is the expected short-term reward function on states and actions.

A policy is in general any set of rules for selecting actions. In this discussion, we consider only policies that are non-randomized and stationary.

A *stationary policy*  $\pi$  is a function  $X \rightarrow A$  mapping the state space into the action space.

We use  $\pi(x)$  to denote the action the policy chooses when in state  $x$ . Broadly speaking, the objective of the agent in the decision problem is to maximize the rewards it receives over time. The agent's current action influences not only the immediate reward it receives in the current state, but also the

rewards it will be able to receive in the future.

There are several ways in which future rewards can be assessed: average rewards, total rewards, and total discounted rewards. In this study, the agent seeks to maximize *total discounted rewards*. This is the simplest case. Furthermore, the learning methods can be used for learning to maximize the total reward or the average reward under certain conditions.

The total amount of discounted reward the agent will receive from time  $t$  is:

$$r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots + \gamma^n r_{t+n} + \dots \quad (\text{eq. 1})$$

where  $r_t$  is the reward received at time  $t$  with  $R(x_t, a_t) = E[r_t | x_t, a_t]$ , and  $\gamma$  is a discount factor. The effect of the discount factor is twofold. The discount factor ensures that the quantity of (eq. 1) is well defined and a reward to be received in the future will be considered less valuable than one received now. This total discounted reward is called the *return*.

In a Markov decision process, the total return depends not only on the state the agent is in at time  $t$  and the policy the agent employs, but also on random factors influencing the state transitions and the rewards. The expected return, however, depends only on the current state and the policy that will be followed. This expected return is written as:

$$V^\pi(x) = E_\pi \left[ \sum_{t=0}^{\infty} R(x_t, \pi(x_t)) \gamma^t | x_0 = x \right]$$

where  $E_\pi$  indicates that the expected value is taken, given that policy  $\pi$  is used to choose actions, and  $\sum_{t=0, \infty}$  is the sum on  $t$  from 0 to infinity. We call  $V^\pi$  the *value function* for policy  $\pi$ .

*Objective:* Find a policy that maximizes the expected return from every state.

The dynamic programming solution to the above problem is obtained by using a recursive functional equation that determines the optimal policy for any state at all stages. This equation is a direct consequence of Bellman's principle of optimality [Bellman61]. It can be derived for the deterministic case as follows. Let

$$V^*(x) = \max_{a_t} \sum_{t=0}^{\infty} R(x_t, a_t) \gamma^t$$

where  $x_0 = x$ . Then

$$V^*(x) = \max_{a_0} \max_{a_t} \left\{ R(x_0, a_0) + \sum_{t=1}^{\infty} R(x_t, a_t) \gamma^t \right\} \quad (\text{eq. 2})$$

The first term in brackets in (eq. 2) does not depend on the second maximization. Thus, (eq. 2) becomes

$$\begin{aligned}
V^*(x) &= \max_{a_0} \left\{ R(x_0, a_0) + \max_{t=1, \dots, \infty} \left[ \sum_{t=1}^{\infty} R(x_t, a_t) \gamma^t \right] \right\} \\
&= \max_{a_0} \left\{ R(x_0, a_0) + \gamma \max_{t=0, \dots, \infty} \sum_{t=0}^{\infty} R(x_{t+1}, a_{t+1}) \gamma^t \right\} \\
&= \max_{a_0} \{ R(x_0, a_0) + W^*(y) \} \quad (eq. 3)
\end{aligned}$$

where  $y$  is the actual next state when taking action  $a_0$  that maximizes (eq. 3) in state  $x$ . In general, however, the transition to a next state  $y$  occurs with probability. Thus, for a Markov decision process, we have (see [Ross83])

$$V^*(x) = \max_a \left\{ R(x, a) + \gamma \sum_y P_{xy}(a) V^*(y) \right\} \quad (eq. 4)$$

This equation is a mathematical form of Bellman's principle of optimality. It states that the maximum expected discounted return for state  $x$  is found by taking the action that maximizes the sum of the expected reward to be received at the present state and the expected maximum discounted return from the state which results from applying the action. We call  $V^*$  the *optimal* value function and any policy whose value function is optimal an *optimal* policy  $\pi^*$ . It is the optimal value function that the agent tries to approximate. Maximizing  $V$  in the *short term* being equivalent to maximizing  $R$  in the *long term* makes dynamic programming particularly attractive. Dynamic programming based on the optimality equation (eq. 4) solves, at least in principle, a variety of important problems. The following sections describe several well-known computational methods of stochastic dynamic programming.

### 3.2 Synchronous Value Iteration

This section describes a successive approximation approach for obtaining the optimal value function--*synchronous value iteration* (SVI). It is a successive approximation method for solving the optimality equation (eq. 4).

In [Ross83], it is shown that when the decision process is in state  $x$ , a policy  $\pi$  that chooses the action that maximizes the right-hand side of the optimality equation (eq. 4)

$$R(x, \pi(x)) + \gamma \sum_y P_{xy}(\pi(x)) V^* = \max_a \left\{ R(x, a) + \gamma \sum_y P_{xy}(a) V^*(y) \right\},$$

is an optimal policy. It follows immediately that we would know the optimal policy should we know the optimal value function. The value iteration method for obtaining the optimal value function is as follows: Let  $V_0(x)$  be any arbitrary bounded function, and define the recursive functional

$$V_n(x) = \max_a \left\{ R(x, a) + \gamma \sum_y P_{xy}(a) V_{n-1}(y) \right\} \quad (\text{eq. 5})$$

for  $n > 1$ . It can be shown [Ross83] that  $V_n(x) \rightarrow V^*(x)$  uniformly in  $x$  as  $n \rightarrow \infty$ , assuming  $R(x, a)$  is bounded for all  $x$  and  $a$ . The method is said to be synchronous because the computation of  $V_n$  depends only on the values of  $V_{n-1}$ .

Thus, to find an optimal policy, we first approximate the optimal value function  $V^*$  by value iteration, *i.e.*, the repeated applications of (eq. 5), and then determine an optimal policy to be the one that selects the action that maximizes the right-hand side of the optimality equation (eq. 4) for each state.

It can be shown that although  $V_n$  is not necessarily closer to  $V^*$  than  $V_{n-1}$  over all states, the maximum error between  $V_n$  and  $V^*$  must decrease. Mathematically, the SVI is a contraction mapping with respect to the  $L_\infty$  norm and has  $V^*$  as its unique fixed point.

### 3.3 Policy Improvement

In addition to the type of approximation for constructing an optimal policy discussed above, *policy improvement* or *policy iteration* is a method of successive approximations in policy space for producing optimal policies [Bellman61]. It is based on the observation that one only needs to know the value function for a given policy in order to improve that policy.

Suppose that policy  $\pi_g$  is some proposed policy, and we wish to know if it is better than the existing policy  $\pi_f$  for which  $V^{\pi_f}$  is known *i.e.* will  $\pi_g$  yield higher expected returns for all states than  $\pi_f$ ? One way to decide if  $\pi_g$  is better than  $\pi_f$  would be to compute  $V^{\pi_g}$  by solving the system of linear equations defined by

$$V^{\pi_g}(x) = R(x, \pi_g(x)) + \gamma \sum_y P_{xy}(\pi_g(x)) V^{\pi_g}(y) \quad (\text{eq. 6})$$

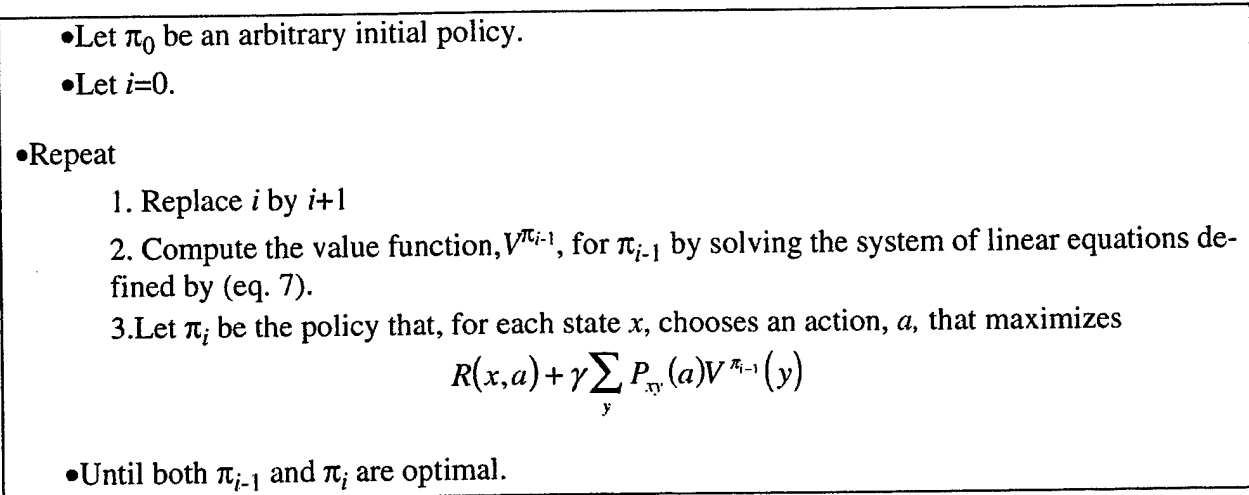
and then compare  $V^{\pi_g}$  with  $V^{\pi_f}$  over all states. However, computing the value function is computationally costly. A more efficient way of doing this that does not require computing  $V^{\pi_g}$  is as follows. Consider the expected return, starting in state  $x$ , that would result from following policy  $\pi_g$  for one step, and then switching to  $\pi_f$  thereafter:

$$V^{\pi_g/\pi_f}(x) = R(x, \pi_g(x)) + \gamma \sum_y P_{xy}(\pi_g(x)) V^{\pi_f}(y) \quad (\text{eq. 7})$$

If  $V^{\pi_g/\pi_f}(x) \leq V^{\pi_f}(x)$  for all states  $x$  with strict inequality for at least one state, then it can be shown that  $\pi_g$  is indeed an improvement over  $\pi_f$ .



The foregoing ideas provide the basis for the policy improvement method. Instead of using (eq. 7) to find out if a proposed policy is an improvement over the current one, the policy improvement method uses it to define a new policy that improves over the existing policy. The new policy is defined to be the one that for each state  $x$ , chooses the action  $\pi_g(x)$  that maximizes the right-hand side of (eq. 7). It can be shown [Ross83] that  $V^{\pi_g}(x) \leq V^{\pi_f}(x)$  for all states, and if  $V^{\pi_g}(x) = V^{\pi_f}(x)$  for all states, then  $V_{\pi_g}(x) = V_{\pi} = V^*$ . The policy improvement method is summarized in Figure 1.



**Figure 1. Policy Improvement Algorithm**

An informal argument for the result goes like this: starting at state  $x$ , it is better to follow  $\pi_g$  for one stage and then follow  $\pi_f$  thereafter than to follow  $\pi_f$  throughout. But by the same token, it is better to follow  $\pi_g$  for one further stage from the state just reached. Repeating this argument shows that it is always better to follow  $\pi_g$  than it is to follow  $\pi_f$ .

### 3.4 Asynchronous Value Iteration and Q-Learning

The classic method of value iteration described in section 3.2 requires that  $V_n$  is computed for all states using the values of  $V_{n-1}$ , and then  $V_{n+1}$  is computed using the values of  $V_n$ , and so on. However, the value iteration method need not be carried out in this way. In fact, the values of individual states can be updated in an arbitrary order. This way of computing  $V^*$  is called *asynchronous value iteration* (AVI) [Bertsekas87, BertsekasTsitsiklis89]. AVI is appealing since the convergence result requires only that the values of all states be updated often enough [Bertsekas Tsitsiklis89, Watkins89], and thus leaves a variety of ways for the use of heuristics to control the process. In practice, this simply implies that whatever strategy is used to choose states whose values are to be updated, no state should suffer from starvation, i.e., no state should ever be prevented from being chosen in the future.

It is important to realize that each individual update of a state's value in AVI does not necessarily make it more accurate as an estimate of the state's true value. However, the estimated value function will converge to the true value function as the total number of updates tends to infinity. It

should also be noted that although the order in which the values of states are updated does not alter the convergence result of AVI, it does influence the rate of convergence.

This form of incremental DP provides a basis on which a variety of learning and planning systems can be developed. For example, a popular TD method, Q-learning [Watkins89], can be viewed as incremental, Monte-Carlo value iteration.

Finally, there are other versions of the value iteration method that are somewhere between the SVI and the total AVI, such as Gauss-Seidel value iteration. In the Gauss-Seidel method, the values are updated one state at a time sweeping through all the states in a sequential manner, with the computation for each state using the up-to-date values of the other states. Like AVI, the order in which the values of states are updated influences the computation. Gauss-Seidel value iteration converges to the true value function under the same conditions under which SVI does.

### 3.5 Discussion

So far we have only discussed the situation where state or action spaces are discrete. When the state or action spaces are continuous, dynamic programming is typically applied by first quantizing the state or action spaces and then applying standard algorithms such as the value iteration method to the quantized state or action spaces as if they were discrete problems.

It is important to realize that although dynamic programming is much more computationally tractable than explicit exhaustive search of all possible state sequences, the nature of the exact computation of dynamic programming is still exhaustive in that it requires the explicit enumeration of all possible states. However, exhaustive computation is very impractical for problems with large number of states, which are typical in real world applications. For this reason dynamic programming has not played a significant role in artificial intelligence research.

In terms of dimensionality of the state space, the practical limitation of dynamic programming can be quantitatively described as follows. As the number of states increases exponentially with increasing dimensionality, so does the complexity of time and space for dynamic programming to produce an optimal solution, and usually reaches a practical limit when the dimension of the state space is about five or six [Fu70]. Approaches that enumerate all possible actions increase similarly in complexity with the dimension of the action space. This problem is also referred to as Bellman's *curse of dimensionality*. It was first discussed by Bellman [Bellman61] and has remained the central issue in dynamic programming since.

In practice, this simply means that the state space may be too large to allocate memory to the entire state space or to update all states repeatedly as are required by the value iteration method. Unless the dimensionality problem can be adequately addressed, the practical utility of dynamic programming remains limited. One of the aims in this report is to look for ways that make dynamic programming more practical and commercially feasible.

## Section 4: Partial States and Learned Features

Foveal vision simultaneously supports a wide field-of-view (FOV), localized high acuity, and high temporal resolution while minimizing sensor data to that which is relevant for concurrent real-time automatic target detection, recognition (ATR), and tracking applications. However, space-variant data acquisition necessitates the development of efficient gaze control mechanisms and rapid schemes for referencing object saliency. This section presents a reinforcement learning method for gaze control, together with simulation studies using an active visual sensor in a simulated gaze control problem, with application to autonomous mobile platforms.

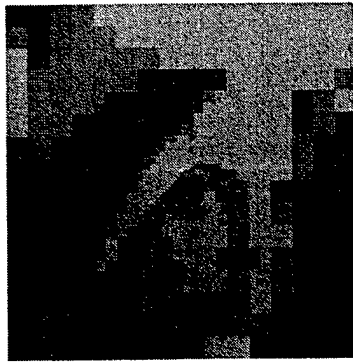
### 4.1 Active Perception as a Partially Observed Markov Decision Process (POMDP)

As developed in the previous section, reinforcement learning is based on dynamic programming, a stochastic optimization procedure guaranteed to converge to an optimal policy only when applied to Markov processes. A stochastic process is said to be Markovian if the future evolution of the process is conditionally independent of the past, given only its present value. In the case of active perception, the agent (active sensor) always has limited sensing resources and cannot delineate the complete world state in a single observation. This presents an the interface which is no longer Markovian in nature. That is, the agent is unable to distinguish between similar world states based solely on the current limited view of the world, and past observations may partially or completely disambiguate these states upon which the next-action decision rests. The problem is characterized by the fact that visual information acquired from a given fixation is in general insufficient to decide where *best* to look next, since different targets may share similar features at similar locations, and others may be out of the current field of view.

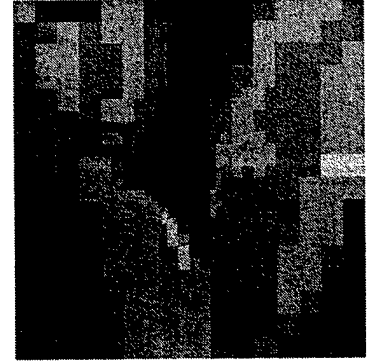
Learning problems in which the optimal action depends on more than the current observation are known as *Partially Observable Markov Decision Problems* (POMDP's). Control of active perception processes (such as gaze control) is always a POMDP since by definition there are relevant parts of the environment which have not been adequately observed at the time the next action is decided. A consequence of the dichotomy between observation and state is *perceptual aliasing*: the incomplete characterization, or misidentification, of the state information required for decision support. Perceptual aliasing may be reduced by the judicious use of past observations. For instance, the limited field of view of our sensor may not include a relevant object which we have seen in an earlier frame. By combining relevant information from past fixations with the current fixation a more comprehensive estimate of world state is attained, and a more informed estimate of potential risks and benefits of looking in various directions can be made. Such approaches to POMDP's are referred to as history-based, and are the focus of our investigations.



*(a) Hat brim fixation*



*(b) Right eye fixation*



*(c) Right shoulder fixation*



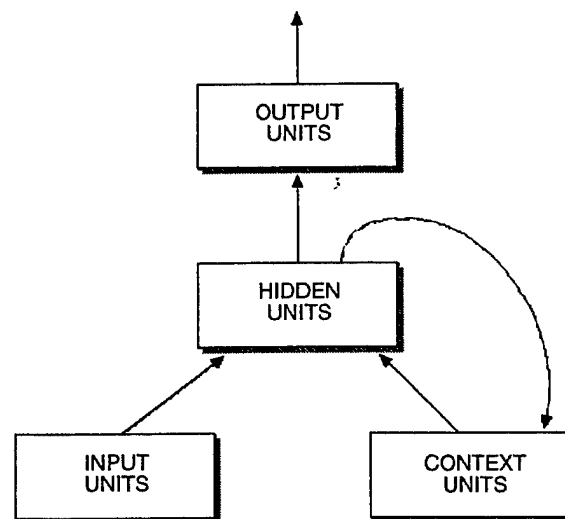
*(d) Integrated perception linking the three fixations*

*Figure 2(a)-(d). Multiple foveal images and integrated perception. Retinotopology: exponential lattice (radix two), four major rings (ie. rexels of five sizes), subdivision factor four (ie. each rexel further subdivided into  $4 \times 4 = 16$  equal size rexels).*

The images upon which gaze control strategies will be determined is of the type illustrated in Figure 2 and described in detail in Bandera90. Such graded resolution biomimetic image acquisition schemes minimize the data bandwidth required to support a given angular field of view and maximum acuity but require effective pointing (gaze control) strategies. Where to look next is a central issue in active perception, is highly task and scenario dependent, and cannot be learned by supervised schemes (except for the simplest of deterministic scenarios which are of limited practical value).

#### 4.2 History Augmentation of Incomplete State Feedback for POMDP's

History-based reinforcement learning may be implemented using look-up tables, sparse coded function approximators or more general neural nets. On large problems, reinforcement learning systems must use compact approximation schemes to represent value functions. There are several approximation schemes, such as CMACs, RBFs, and other neural networks, that are capable of the real-time performance essential in many time-critical applications. The particular function approximator we use is the Elman neural network. Neural networks are general approximators and are capable of implementing any function to any desired degree of accuracy given sufficient resources. In addition, there exist learning techniques well suited for recurrent networks that are required for many tasks involving time, including the gaze control problem we study here.



*Figure 3. An Elman Net*

Recurrent neural networks, such as Elman networks, provide a way to construct useful history features that are essential for solving partially observable Markov decision problems. As shown in Figure 3, the input to an Elman network consists of two parts: input units and context units. The context units carry feedback activations resulted from the network state at the previous time step. That is, the context units at time step  $t$  are copies of the hidden units at time step  $t-1$ . As a memory structure, the context units remember an aggregate of previous network states. As a result, the behavior of the network depends on past as well as current inputs.

Adjustable weights in an Elman network are those between the input units and the hidden units, those between the context units and the hidden units, and finally those between the hidden units and the output units. To predict utility values correctly, the recurrent network will be forced to discover historical features that permit the network to properly assign utility values to inputs displaying the same perception.

The recurrent network is trained using the SARSA algorithm in conjunction with back-propagation. SARSA is a based multi-step learning technique [PengWilliams94,96, RummryNarajan94]. The SARSA method has shown dramatic performance improvement over Q learning on all the tasks that have been examined so far [PengWilliam94, Rummry Narajan94,Sutton95]. In Q-learning, error correction is made only to the Q value estimate of current state-action pair. To make correction to earlier actions, many trials will have to take place. In contrast,  $Q(\lambda)$  based techniques use a trace mechanism, which is a memory structure, in such a way that error correction can be made not only to the Q value estimate of current state-action pair, but also to that of earlier state-action pairs along the trace. This in a sense amounts to efficient use of experience and parallels only to that of EBL [Mitchell86]. More recently, Sutton [Sutton96] argues that in an environment where learning takes place continuously, as it should, SARSA has the advantage of following the policy it is actually estimating, whereas Q-learning estimates a policy that it does not follow. In terms of average reward received per trial, SARSA performs significantly better than Q learning. The SARSA algorithm is described in the box below, where  $W$  is the weight vector,  $Q$ , parameterized by  $W$ , is the current approximation to action values,  $r$  is the immediate reward,  $\gamma$  is the discounted factor,  $\alpha$  is the learning rate, and  $\lambda$  is a procedure (meta) parameter that controls credit distribution over time.

1. Reset all eligibilities,  $e_0=0$
2.  $t = 0$
3. Select action,  $a_t$ .
4. If  $t > 0$  then  $W_t = W_{t-1} + \alpha (r_{t-q} + \gamma Q_t - Q_{t-1}) e_{t-1}$
5. Calculate  $\Delta_w Q_t$  with respect to. selected action  $a_t$  only.
6.  $e_t = \Delta_w Q_t + \gamma \lambda e_{t-1}$
7. Perform action  $a_t$ , and receive short-term reward  $r_t$ .
8. If trial has not ended, replace  $t$  by  $t+1$  and go to step 3.

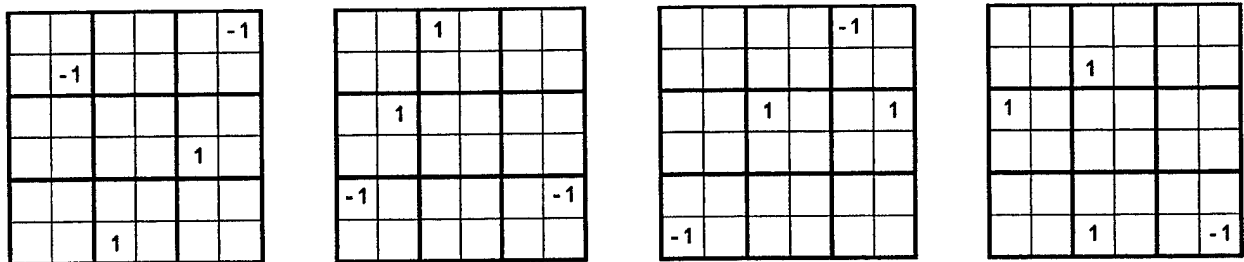
### *The SARSA Algorithm*

#### **4.3 A Token Gaze Control Problem**

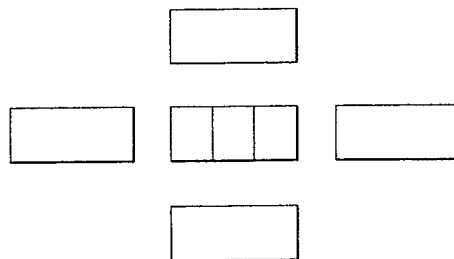
One of the main attributes of the foveal vision gaze problem we are interested in is that multiple fixation points, and the accumulation/integration of relevant visual information acquired from each

gaze, will be required to classify a target to within some confidence threshold. This requires the scale-space volume defined by the discriminant target features to be greater than the scale-space volume supported by the foveal retinotopology. Figure 3 shows the simulated recognition problem used for the experiments reported in this section.

Each graph in Figure 4 below represents a region-of-interest where a potential target has been pre-attentively located. Each target is represented by four features (1 or -1) placed in various locations in the ROI. It should be noted that both classification features and their locations (highlighted boxes) are important for determining a target. However, the precise feature position in each highlighted box is unimportant. The foveal agent has limited sensing capabilities. The objective is for the agent to interrogate and extract spatial patterns in a scale space fashion with a minimum amount of interrogations (saccades) and computation.



**Figure 4. Simulated Recognition Problem**



**Figure 5. Foveal sensor with high acuity center and reduced acuity surround**

The foveal agent is equipped with a sensor with five receptive fields, shown in Figure 5, that it can manipulate using a given set of four control actions. These actions can steer the sensor in four compass directions: *left*, *right*, *up*, and *down*. The center receptive field has a higher resolution capable of encoding the type of a feature and its precise location. The peripheral sensing device has a low resolution and can only encode presence or absence of a feature within its receptive field. For simplicity we will assume that only one feature type, -1 or 1, may be present in any highlighted box.

One can pose gaze control for the ATR task as a learning problem in many different ways. In this report gaze control for active vision is formulated as a reinforcement learning problem. The short-

term reward for each action is zero except the one that leads to a target recognition. An alternative reward structure is on-line reinforcement, which accelerates convergence of the value function [Bandera96]. The performance measure is total discounted reward received over time. Thus, the optimal performance according to this metric is for the agent to choose the shortest gaze (saccades) sequence required for a target recognition.

#### 4.4 Simulation Results

This section reports the experimental results of the SARSA algorithm combined with recurrent networks for learning optimal gaze control in the foveal ATR problem described in Section 4.3. Through these experiments, we expect to gain more insight into how well these learning systems work under different conditions. The experiment consisted of a series of trials. A trial here is a sequence of gazing fixations. A fixation sequence begins with the active sensor pointing at a starting location within the region-of-interest. It ends when all four features are detected and classified.

After some experimentation, we settled on an Elman network with ten input units, six hidden units, six context units, and one output unit that encodes action values for a given state-action pair. Note that the number of hidden units could have been determined via cross-validation. The input units are as follows:

- three units: These are the inputs to the center sensing device. One encodes information about the feature being detected. Two units encode the location of the feature.
- four units: Each represents a feature being detected by each of the peripheral sensors.
- two units: These encode 4 control actions.
- one unit: The bias unit is always on.

The weights are initialized to lie between -0.1 and 0.1 and updated after each gaze. This is in direct analogy to the typical weight update procedure in neural networks where weights are updated according to the stochastic gradient or incremental procedure instead of the total gradient rule [LeCun91]. That is, updates take place after each presentation of a single exemplar without averaging over the whole training set. Both empirical and theoretical studies show that the stochastic gradient rule converges significantly faster than the total gradient rule, especially when training set contains redundant information.

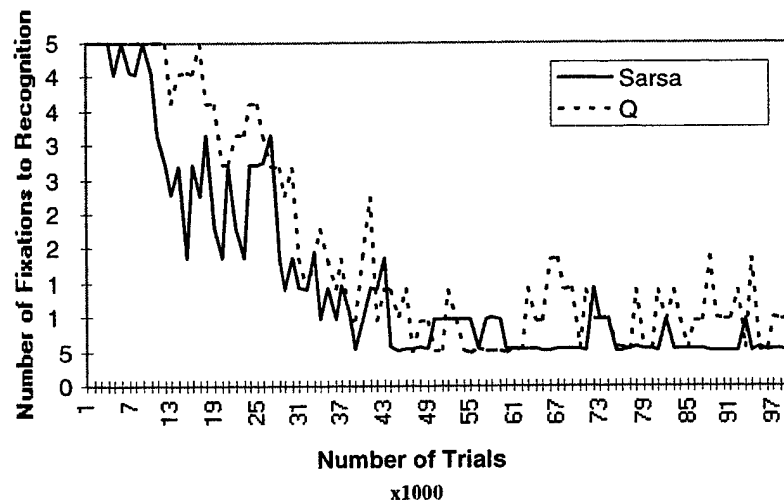
There are several procedural (meta) parameters that have to be determined experimentally: the discounted factor, the weighting parameter, and the learning rate. Values for these parameters were 0.9, 0.3, and 0.5 respectively. The experimental tests showed that good performance can be obtained under wide range of these parameter values. In addition, during training the agent employs an  $\epsilon$ -greedy policy to ensure sufficient exploration. An  $\epsilon$ -greedy policy is one that, at each state, chooses greedy actions  $(1 - \epsilon)$  percent of time and random actions  $\epsilon$  percent of time. In the experiments reported within this paper,  $\epsilon$  was chosen to be 0.15 throughout. Convergence to the optimal policy in each case was attained within 15,000 trials.

An additional experiment, whose detail we omit here, was carried out in which some feature locations in highlighted boxes were perturbed. The intent here was to model noise sensor reading and



possible distortion in a perception process under real-world conditions, and to evaluate the robustness of the learned gaze control policy. The result showed that the same gaze sequences were indeed recommended by the control policy, illustrating the generalizing capability of the learning system.

To view these results, the learning curve for this system was graphed in Figure 6. As a comparison, the performance of Q-learning as a function of time was also graphed. The learning curves were an average over 10 runs. Both algorithms used the same sets of initial random weights. Figure 5 shows clearly the superior performance of SARSA over Q-learning. Furthermore, SARSA exhibits more stable behavior than Q-learning despite the fact that both employed the same  $\epsilon$ -greedy policy. The result is consistent with prior empirical studies by others. In addition, to the best of the authors' knowledge, this is the first documented empirical study that combines SARSA with recurrent neural networks.



*Figure 6. Gaze control policies based on SARSA and Q learning*

#### 4.5 Discussion

We have presented a history-based reinforcement learning method for the control of active vision gazing. The associated simulation results support the judgement that reinforcement learning is a feasible learning instrument for use in gaze control in autonomous or semi-autonomous platforms.

Reinforcement learning, in conjunction with relevant history features also learned by reward and punishment, is capable of solving active perception tasks where perceptual aliasing is prevalent and poses significant challenges to traditional reinforcement learning methods. Our results show that for a simple (token) gaze control problem, the optimal strategy in a partially observable Markov environment can be successfully learned using this approach without training periods of undue length. In addition, comparing reinforcement learning methods, it was found that SARSA offers significant improvements in learning speed and reliability over Q-learning in this environment.

History-based reinforcement learning for gaze control provides large advantages over non-learning control methods in terms of adaptability, efficiency, reliability, and apparent commercial feasibility. Having been shown feasible for a token problem with relatively few states, scalability to more practical gaze control problems must be considered. This is the subject of the next section.

## Section 5: Scaling Up - Partial States and Fixed Features

To approach optimality, a decision process must be aware of its current state at each decision event. POMDPs are decision processes in which the current observation does not specify the process state completely. History-based approaches to reinforcement learning in POMDPs attempt to narrow the gap between the available and the required data by mapping the past observation sequence into a smaller dataset of history features. These history features augment the current observation, or features extracted from the current observation, together forming a *quasi-state* for purposes of defining evaluation functions and credit assignment in reinforcement learning.

Let  $x \in X$  indicate a state of the process and  $s(x) \in S$  a quasi-state description of the process (concatenation of current observation features and history features) while in state  $x$ . If no information from past or current observations which is relevant to current or future optimal decisions is lost in the process of compressing the full observation sequence into the selected features, the maximum expected discounted return (as in eqn. 4) will satisfy

$$V^*(s(x)) = V^*(x)$$

and, by the principle of optimality, the optimal policy  $\pi^*$  can be determined from the evaluation function  $V^*$  defined over the set of quasi-states  $s \in S$  no less than over the actual state space  $x \in X$ .

One choice of quasi-state which clearly guarantees that there is no loss of relevant information is to use the entire past and current observation sequence as the set of features. While guaranteeing eventual optimality, this strategy is practical only in small problems in which the size of this dataset is acceptably limited. In the token detection problem developed in the previous section, each past observation can be encoded in 2 bytes (3 bits for the high acuity zone, 7 to encode all the low acuity zones, and 6 for position of the sensor). There are at most 36 past sensor positions, thus the entire history, however long, may be stored in a data array no bigger than 72 bytes.

For larger problems some loss of relevant information is inevitable. The approach of the previous section was to use an Elman net both to iterate towards optimal features and to use its current quasi-state to define evaluation functions and rewards. If the compression of observation sequence to feature set is too great, it may be anticipated that this process will be slow. Additionally, if the features include a memory trace spanning many observations, the Elman net will (at best) exhibit slow convergence toward appropriate weight values.

The fundamental problem is that learning a task-efficient unconstrained lossy state data compression mapping *simultaneous with learning accurate evaluation functions* based on the quasi-states is difficult in higher dimensional spaces. Unguided by domain knowledge, the initial feature selection is likely to be so poor that the evaluation functions learned are of little value in refining the feature selection (Elman context unit weights). Thus the entire learning process resembles a random walk through the high-dimensional cross-space of features and evaluation functions.

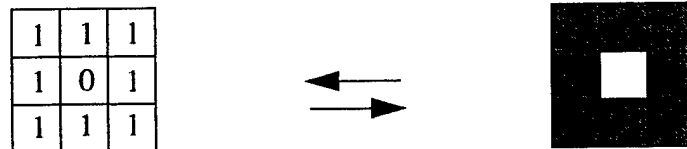
For such problems it may be possible to define useful fixed features based on prior domain knowledge. For instance, if we are using a camera with limited field of view to search for ran-

domly dispersed squares in a large static field of regard we should remember where it is that right angles are observed. Other edge patterns are not relevant to this task and need not be retained. The only other feature useful to encode is the location of each observation, so we do not revisit mapped regions of the field of regard. Given these features characterizing the current observation and the observational history, we have all information from present and past observations relevant to the decision problem. While good feature selection strategies such as this are often easily derived from domain knowledge, it might be a very long training period indeed before a general learning algorithm deprived of domain knowledge and driven only by rewards and punishments settles on acceptable feature definitions.

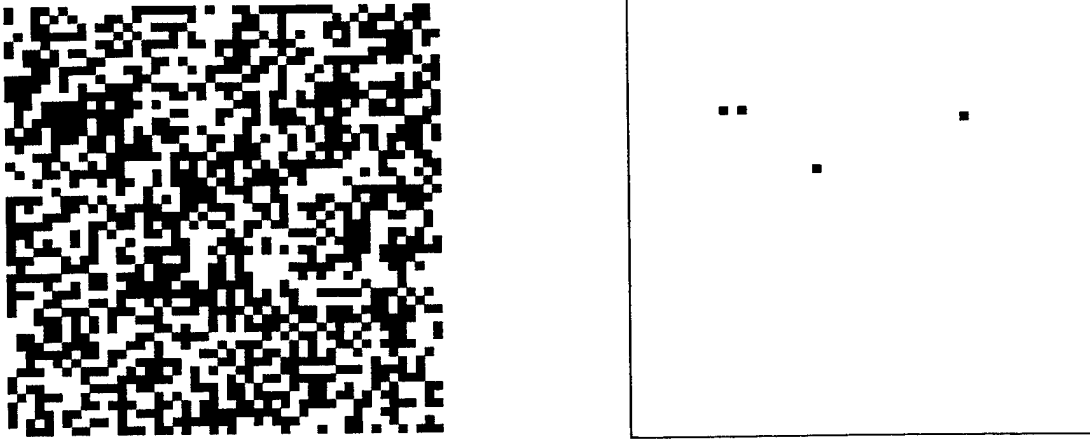
### 5.1 Problem Description

In this subsection a scaled-up gaze control problem is posed. The agent is equipped with an active vision sensor that it can orient using a given repertoire of control actions. The goal is a gaze control strategy that will detect a specific pattern (the target) efficiently, i.e. with least cost. Cost may be scored as time-to-target (number of frames of data processed on average before a target is successfully found), travel-to-target (minimum total angular slew of the optical axis), or some combination of these measures. The environment, or set of pixels which may be searched, is modeled as a square two dimensional lattice of binary white noise. The target is defined as a specific 3x3 binary pattern, and the control actions limited to a few North-South or East-West movements of the optical axis of the system. The details of this problem are specified in the following paragraphs.

The environment that the agent is to search is a large planar lattice of square pixels. The binary value (0 or 1) of the pixel is the result of repeated Bernoulli trials with equal probabilities  $P(0) = P(1) = 0.5$ , all pixel values are independent. Thus the whole scene can be described as white noise with binary independent identically distributed (BIID) pixels. The specific target that the agent will search for is defined by the 3x3 binary template in Figure 7(a). Note that this target has a mean value of  $8/9$  as opposed to the expected background of  $1/2$ . The difference in this statistic will be a useful cue when the target is viewed at low resolution. Note also that the targets are not superposed on the background after the background has been generated, they are simply part of the BIID scene. Thus the probability of a given pixel location being the center of a target is exactly  $(1/2)^9$ . A 50x50 scene is shown as Figure 7(b), with the centers of the four targets found in this scene indicated.



*Figure 7(a). Numerical and graphical representation of the target template*



*Figure 7(b). A 50x50 sample environment and its target distribution*

The signal to noise ratio (SNR) is next computed. For the problem as described, the signal  $s_{ij}$  consists of a random set of targets (as shown in Figure 7a) against a white (0 valued) background, and the noise  $n_{ij}$  is the remainder of the BIID scene  $x_{ij}$  as in Figure 7(b):

$$x_{ij} = s_{ij} + n_{ij}$$

The mean and second moment of the signal and the noise  $n_{ij}$  may be calculated combinatorially. Expanding the expected values for this stationary binary random field,

$$Es_{ij} = Es_{ij}^2 = 1 \times P(s_{ij}=1)$$

$$En_{ij} = En_{ij}^2 = 1 \times P(n_{ij}=1)$$

The event  $\{s_{ij}=1\}$  that the signal pixel at  $(i,j)$  equals one (is black) is the event that the center pixel of a target happens to be one of the eight nearest neighbors of the point  $(i,j)$ . Consider a 5x5 array of pixels centered at  $(i,j)$ . To evaluate the probability of this event we count the number of primitive outcomes associated with it. First consider the number of outcomes consistent with a target northwest (NW) of  $(i,j)$ , i.e. centered at the location  $(i-1,j+1)$ . There are  $2^{25-9} = 2^{16}$  such distinct binary 5x5 arrays. There are  $2^{16}$  outcomes with a target NE, of which  $2^{10}$  have already been counted (contain both NW and NE targets). Thus the number of primitive outcomes with targets NW or NE is  $2^{16} + (2^{16} - 2^{10})$ . Continuing, there are  $2^{16}$  outcomes with SE targets, of which  $2^{10}$  have already been counted in each of the two previous cases, so  $2^{16} - 2 \times 2^{10} + 2^4$  primitive outcomes must be added (the last term corrects for counting duplicates twice since there are  $2^4$  outcomes with targets in all three of the NW, NE and SE corners). The SW case adds  $2^{16} - 3 \times 2^{10} + 3 \times 2^4 - 1$  primitive outcomes. Adding, there are

$$4 \times 2^{16} - 4 \times 2^{10} + 4 \times 2^4 - 1 = 258,111$$

primitive outcomes of 5x5 binary arrays with targets to the NW, NE, SE or SW of the central

pixel. The set of binary arrays with targets centered N, E, S, or W, can be constructed by reordering the pixel locations of the set of those centered NW, NE, SE and SW, so the total number of primitive outcomes with a target centered at one of the eight nearest neighbors of  $(i,j)$  is double the above, and

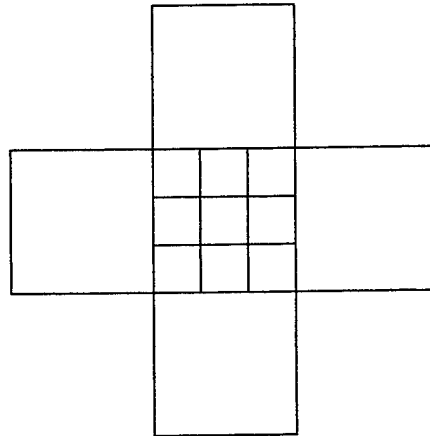
$$P(s_{ij}=1) = (2 \times 258,111) / 2^{25} = 0.01538$$

Note that the sample environment depicted in Figure 7(b) is consistent with this value, since the four target instances found comprise 29 black pixels out of 2500. Substituting (4) into (2) we compute the SNR to be:

$$\begin{aligned} SNR &= \frac{\sigma_s^2}{\sigma_n^2} = \frac{Es_{ij}^2 - (Es_{ij})^2}{En_{ij}^2 - (En_{ij})^2} \\ &= \frac{0.01538 - 0.01538^2}{0.5 - 0.5^2} = 0.06057 \end{aligned}$$

Thus the SNR is thus -12.2 dB, a relatively low value for reliable target detection.

We experimented with both uniform resolution and multi-resolution sensors. The former contains a  $3 \times 3$  array of sensor lattice sites (pixels), the latter consisted of a  $3 \times 3$  fovea surrounded by a ring of resolution cells or "rexels" each 9 pixels large as depicted in Figure 8. Each of the rexels produces a single output value, the sum (or average) of the 9 pixel values within its receptive field.



**Figure 8. 2-ringed undivided exponential foveal sensor (radix 3) with partial coverage of outer ring**

Note that the presence of a target exactly registered in one of the low-resolution rexels outside the fovea is signalled by a high rixel value, 8 out of a maximum of 9 (or  $8/9$  if the average rather than total value is reported by these rexels). This value does not guarantee target present, however, since there are 8 non-target patterns that would yield the same rixel value.

## 5.2 Actions

Three different sets of control action were used to simulate the agent's capacity to steer the visual sensor. These actions translate the fixation point, the location in the scene imaged at the center of the fovea, as follows:

**Set A:** Move N, E, S or W a distance of 1-3 pixels.

**Set B:** Move N, E, S or W a distance of 1-5 pixels.

**Set C:** Move along the 8 principal compass directions 1-3 pixels

A cost is associated with each action that the agent performs. The goal is to minimize the discounted cumulative cost to first detection of an instance of the target. Specifically, the cost is :

$$J = \sum_{i=0}^{n-1} \gamma^i (C_1 |a(i)| + C_2)$$

where  $a(i)$  is the action taken at the  $i$ th step,  $|a(i)|$  is the distance moved by the fixation point in pixels,  $C_1$  and  $C_2$  are constants defining the relative importance of distance and time, and  $n$  is the number of time steps to first target detection. This performance index reflects our interest in minimizing time to target acquisition. The first term models time to slew the optical head, and the second models fixed dwell time to process the new fixation.

Initially, the agent's visual sensor is aligned to a random location in the scene. The agent receives a frame of data from the sensor, and selects an action. Upon execution of that action and receipt of the next frame of data, either a penalty or a scalar reward value is received depending on whether a target has been registered. This perception-action sequence continues until a target is detected or the search is declared terminated with failure due to hitting hard time or space bounds. In any of these cases, the trial is declared complete and its results recorded.

The gaze control policy (action selection strategy) will be modified and improved as the agent continues interacting with the environment. The goal of the agent is to converge toward an optimal policy minimizing the cost function  $J$ .

## 5.3 Reference Policies

Before considering reinforcement learning approaches to this problem, a pair of comparison or reference policies not based on learning are described. The performance of these policies will serve as a basis of comparison for the reinforcement learning results.

To establish a performance floor, random walk experiments using the action sets A, B and C

described in Section 5.2, with a uniform resolution  $3 \times 3$  sensor, were performed. For each trial, the sensor is initially fixated at a random location in the environment and the agent randomly selects an action from the selected action set with uniform probabilities, repeating until the target is detected (registered with the sensor) or the limit of saccades is reached. When a target is found or the saccade limit reached, the trial is complete. The results of 100 trials averaged for each of the action sets is shown in Table 1.

**Table 1: Unconstrained random policy performance**

Results of 100 Trials	Distance to target (pixels)	Time to target (fixations)
Set A	1236.77	616.90
Set B	1351.82	450.28
Set C	1182.20	589.62

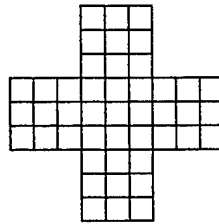
A second, and better, non-learning-based policy was employed in order to provide a stringent baseline against which to evaluate the performance of the reinforcement learning strategies. The policy is based on perfect recall of all past perceptual data. Actions are chosen to maximize the true probability of immediate one-step target detection, with no exploration, and this policy is therefore called the *greedy-immediate policy*. It assumes the agent possesses unlimited memory, remembering all past perceptions exactly. Given a perception, for every pixel  $(i,j)$  in the entire scene, the value of  $P_{TC}(i,j)$ , the probability that pixel is a target center given the complete past sequence of perceptions, is evaluated. Initially all such probabilities are set at  $2^{-9}$  since all  $512 \times 3 \times 3$  binary patterns are equally likely everywhere. Upon each new perception, after all pixel target probabilities are updated, the agent will choose the action that will saccade to the fixation point with the maximum probability of being a target center among all pixels reachable in a single move. If there are several such best one-step candidate fixation points, one is selected at random.

Define the set of pixels  $Y = \{(i,j)\}$  contained in a given perception, augmented by their immediate nearest neighbors, as the update set  $U = \{(i+k,j+l) : (i,j) \in Y; |k|, |l| < 2\}$ . Only pixels  $(i,j) \in U$  require updating, since the new perception contains no data relevant to the remaining  $P_{TC}(i,j)$  values. For each  $(i,j) \in U$  the probability  $P_{TC}(i,j)$  that pixel location  $(i,j)$  is the center of a target is updated as follows. Let  $P_B(i,j)$  be the probability that pixel  $(i,j)$  is black and  $P_W(i,j) = 1 - P_B(i,j)$  the probability it is white. After a given perception, every pixel in the scene is in one of three states: known to be black  $P_B(i,j) = 1$ , known to be white  $P_W(i,j) = 1$  or as yet unobserved  $P_B(i,j) = P_W(i,j) = 1/2$ . For each  $(i,j) \in U$  compute  $P_{TC}(i,j)$  as the nine-way product of  $P_W(i,j)$  and  $\{P_B(i+k,j+l); |k|, |l| < 2\}$ . Thus for instance if  $(i,j)$  is known to be black, or any of the eight-nearest-neighbors of  $(i,j)$  are known to be white, the location  $(i,j)$  cannot be a target center and  $P_{TC}(i,j) = 0$ . In general if  $m$  of the nine pixels centered at  $(i,j)$  are known to be of the right color for a target (white for  $(i,j)$ , black for the remaining pixels), none of the wrong color, and the remaining ones unknown,  $P_{TC}(i,j) = 2^{m-9}$ .



The performance of the greedy-immediate policy was explored in a second set of numerical experiments. Action set A (move 1, 2 or 3 pixels N, E, S or W) was selected for these experiments. Two different perception geometries were simulated: a small 3×3-pixel uniform resolution sensor, and a wide FOV sensor containing five such arrays in the cross pattern shown in Figure 9.

Comparing Table 1 and Table 2, as expected the greedy-immediate policy performs much better than the unconstrained random policy. Indeed, these two non-learning-based heuristic policies, one a random policy totally lacking in goal-directed strategy, and the other the presumably near-optimal, may be anticipated to bracket the performance of any interesting learning algorithm.



**Figure 9.** Wide FOV uniform resolution sensor geometry

**Table 2: Greedy-immediate policy performance**

Results of 100 Trials	Distance to target (pixels)	Time to target (fixations)
3×3 uniform resolution sensor	336.16	168.94
Wide FOV sensor	338.48	167.79

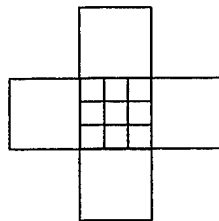
#### 5.4 Fixed Feature Algorithms Tested

If practical necessity persuades us to abandon guarantees of convergence based upon the Markovian assumption, a partial state derived from current and history features may be used. "Perception-state" spaces, or feature-based compressed descriptions of the current observation, may be defined in various ways. For the 3×3 uniform resolution sensor, two perception-state spaces are proposed. The first one, designated *frame space*, is the limiting uncompressed case and consists of the complete current perception. The second perception-state space for the 3×3 sensor is constructed by extracting and then enumerating certain features of the current frame, and the number of these perception-states is much less than that for frame space. This space will be referred to as *feature space*. It is assumed that the agent can preprocess perceptual data to extract specified features. A search is conducted along the 4 compass directions to determine if there is a 1/3 partial match, 2/3 partial match or no match of the target template. These four values constitute the feature set. Since there are 3 possibilities for each of the 4 directions, the total number of states in feature space is  $3^4 = 81$ .

Compass Direction :	East	South	West	North
2/3 match of target :				
1/3 match of target :				
No match of target :	None of Above	None of Above	None of Above	None of Above

**Figure 10.** Feature components for the construction of the feature space (81 possible states in total). “?” pixel stands for don’t care condition

A modified version of the wide FOV sensor shown previously in Figure 9 was also considered. In order to determine the value of having a lower-resolution set of rexels around the fovea, the nine pixels in each of the four peripheral 3×3 regions are combined to a single rexel outputting a single value corresponding to the number of black pixels within its scope. The resulting sensor geometry is shown in Figure 11.

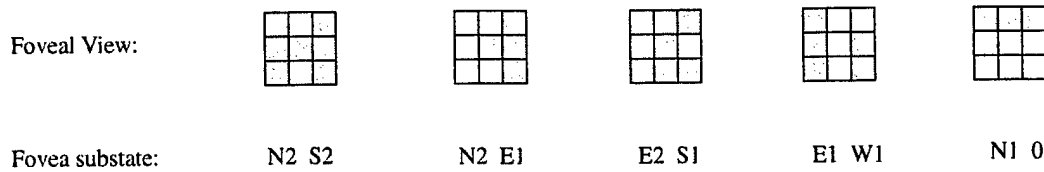


**Figure 11.** Modified version of the wide FOV sensor with a 3×3 fovea and 4 rexels along the four principal compass directions

Note that the original wide FOV sensor has  $2^{45} = 3.52 \times 10^{13}$  distinct states, too many rows for a practical Q-value look-up table. The modified sensor of Figure 11 has 10 distinct values 0-9 for each of the four peripheral rexels and 2 values for each of the 9 foveal pixels for a total of  $2^9 \times 10^4 = 5.12 \times 10^5$  states (rows), a reduction of more than seven orders of magnitude, but still too many. A perception-state space for this sensor which is similar to the feature space described previously is constructed as follows:

(1) The 9 foveal pixels are preprocessed to determine the degree of partial match to the target in the N, E, S and W compass directions. The direction and magnitude (0, 1/3 match, 2/3 match) of

the two best matches are stored in order as the foveal substate. 29 distinguishable foveal substates are obtained in this fashion. Examples of these substates are shown in Figure 12.

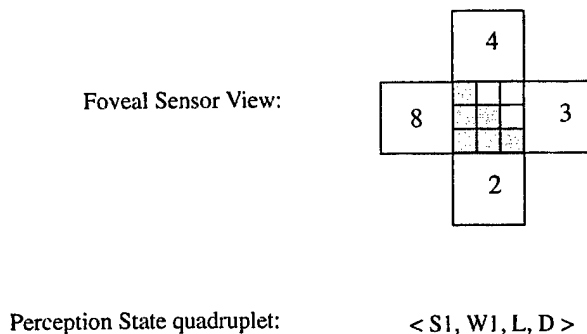


**Figure 12.** *Examples of some of the fovea substates. The letters stands for the direction of the match, the number stands for the match degree, (2)/3 match, (1)/3 match and 0 for no match*

(2) The peripheral rexel values are quantized into the following four sets depending upon their value (number of black pixels):

- 7-9 --->D (very Dark)
- 5-6 --->d (dark)
- 3-4 --->l (light)
- 0-2 --->L (very Light)

The quantized values of the two rexels located in the directions of the two best foveal matches included in step 1 above are appended to the foveal substates to complete the construction of the perception-state. An example is shown in Figure 13. Since there are 4 quanta for each of the two corresponding rexels, the total number of distinct perception-states is  $29 \times 4 \times 4 = 464$ . This is a manageable number of rows for a Q-value look-up table.



**Figure 13.** *Example of a perception-state. The number represents rexel value*

This mapping is certainly lossy, with the  $5.12 \times 10^5$  distinct states supplying more information of use to target-seeking than the 464 perception-states. Previous experiments showed, however, that effective actions tend to move the fixation point in the direction of best match, that is, the most likely direction to find a near-by target. Knowledge of the rexel value in such a direction is some-

times useful, for instance if there is a 2/3 foveal match in the N direction but the rexel in that direction has a value of 0, 1 or 2 there cannot be a target found by moving one pixel to the north. Most of the time the rexel value will be light or dark, but not very light or very dark, and thus yield a weak cue only. Thus it is anticipated that the performance of this sensor will be some improvement over the performance of the 3×3 sensor not perceiving this cue, but not a great improvement.

To complete the partial state description, history features must be selected to augment the perception-state. In the present case, the most important history feature is the list of locations of all nearby previous observations, or *fixation-window*. The agent keeps a W×W bit sliding window centered at the current location marking those nearby locations that have been visited since they come into the fixation-window range. This will be used to constrain fruitless revisits to regions already adequately explored and found to be empty of targets. While other history features are relevant, such as previously acquired partial match scores, these are not strong determinants of optimal or near-optimal policies and were not considered further.

Feedback signals are generated as soon as the agent takes an action. A unit positive reward is assigned only when the agent has successfully detected a target at full resolution (ie. in its fovea). A unit negative reward or punishment associated with the cost function described in Section 5.2 will be applied if the current action fails to yield a target detection.

The discount factor  $\gamma$  is used to determine to what degree rewards in the future effect the current policy choices. In this problem, due to the unavailability of detailed history, the task of target detection is similar to a reflexive behavior, less like planned behavior along a path. These reflexive behaviors constitute quick reactive saccades in response to the most recent feature space value. Since previous quasi-states have little correlation to future successes given the present, a low discount factor in the range 0.1-0.2 was used.

The Q-learning algorithm implemented here selects action  $a$  from perception-state  $s$  with the best Q-value  $Q(s,a)$ . In order to drive sufficient exploration of all the state-action pairs and to avoid trapping inside infinitely repeated inferior action loops without new policy choices, the actions in the reported experiments are selected probabilistically based on Q-values using a Boltzmann distribution. Given a perception-state  $s$ , the corresponding action  $a$  is chosen with the probability:

$$P(a) = \frac{e^{Q(s,a)/T}}{\sum_{a \in A} e^{Q(s,a)/T}}$$

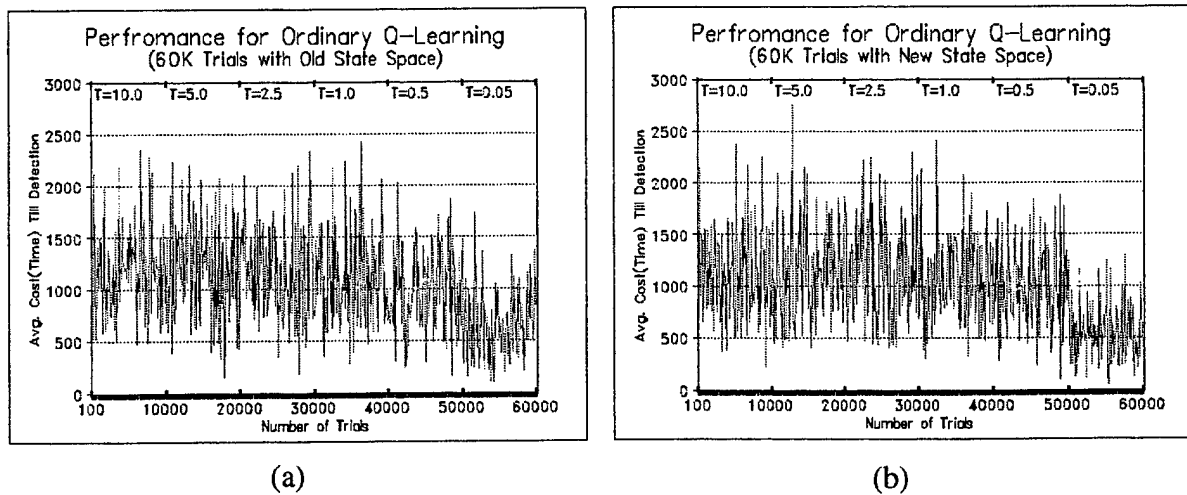
where  $A$  is the set of all available actions (Action set  $A$  in this token problem) and  $T$  is the computational temperature parameter that controls the trade-off between exploitation and exploration. The action with the largest Q-value is more likely to be selected, while actions other than that one also have a chance to be selected.  $T$  is decreased over time by a cool-down function as exploration gives way to exploitation.

## 5.5 Results

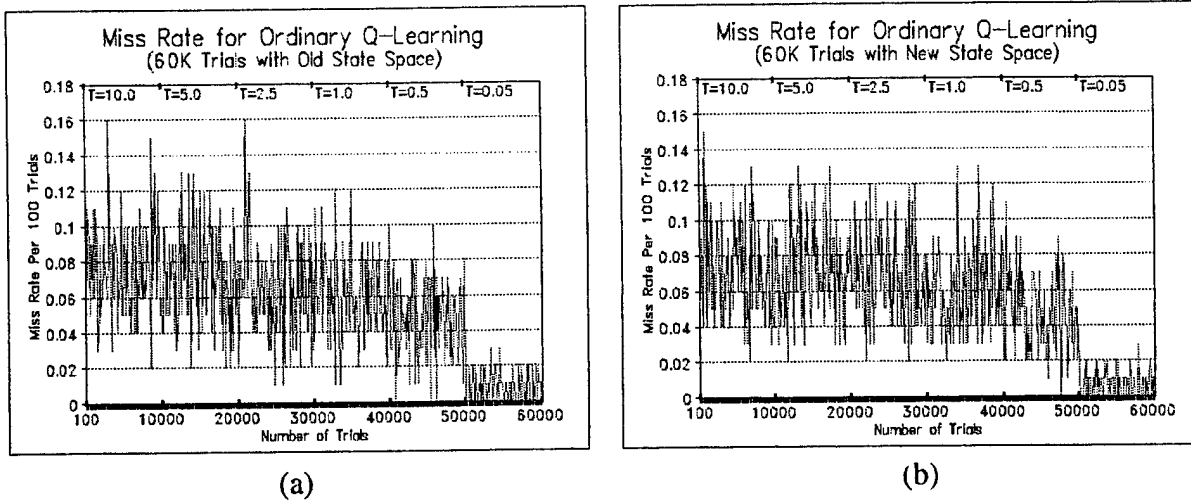
The experiments are divided into two groups. The first uses no memory features, and is referred to as *standard Q-learning*. The second, referred to as the *fixation-window approach*, employs the fixation-window history features described in Section 5.4. Results from the two approaches will be compared. Moreover, both perception-state spaces (frame space and feature space) will be used to better understand the relationship between learning efficiency and feature selection in the current environment.

Performance is measured by averaging the total penalty for each trial. The miss rate is calculated by counting the number of trials which failed to locate a valid instance of the target within 2000 frames together with all those trials whose fixation trajectories crossed the spatial limits of the environment. Both time to target and miss rate indexes are tabulated.

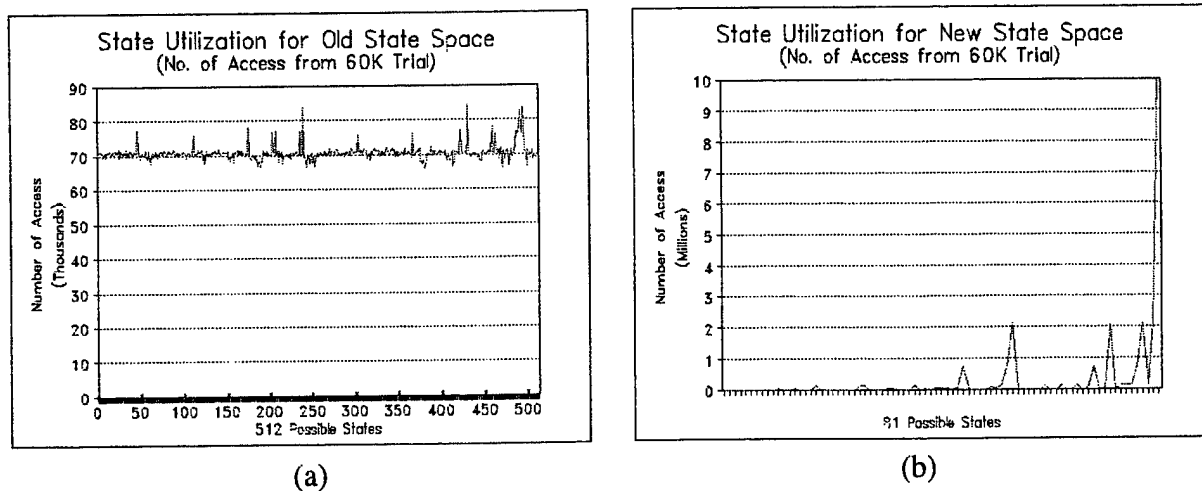
Standard Q-learning will be considered first, using both frame and feature spaces. A relatively large number of trials (60,000) were used for training to guarantee adequate exploration of the state-action pairs. In this setup there are  $512 \times 12$  Q-value table entries for the frame space runs, and  $81 \times 12$  in the feature space case. The computational temperature  $T$  (tabulated on the top of each chart) is decreased step-wise as indicated. Please note that on some graph legends the term *Old State Space* is used synonymously with *frame space*, and *New State Space* with *feature space*. The results from the experiments are summarized in the following figures.



**Figure 14.** Performance (average cost  $J$  over 10 trials by standard Q-learning for 60K trials (a) frame space (b) feature space



**Figure 15.** Miss rate by ordinary Q-Learning for 60K trials with (a) frame space (b) feature space



**Figure 16.** State utilization (number of access to the states) for (a) Frame space. All of the 512 states are utilized (b) Feature space. Only 34 out of the 81 states are utilized, state 81 registered the highest number of access since it represents the no-match condition for all directions

It is seen that results from experiments using the two perception-state spaces produce similar performance after learning. The resulting policies (when  $T=0.05$ ) demonstrate an improved performance over unconstrained random behavior, the mean total time is reduced from around 1000 steps to around 500 steps. On the other hand, due to the dynamic environments (a new scene for each trial) and the violation of the Markov assumption, Q-learning is somewhat slow to converge:

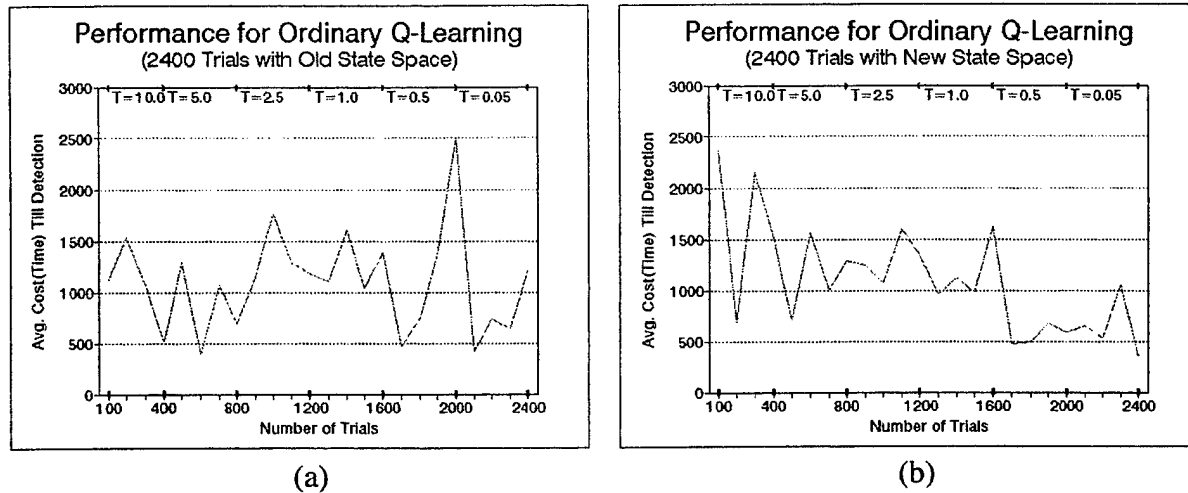
the resulting policy can have a performance of 100 steps for some cases while as much as around 1500 steps for others. The same event is observed by plotting of the miss rate: though the agent is more capable of detecting targets (after learning mean miss rate reduced from 0.07 to 0.01), the resulting policy also saw the miss rate vary between 0 and 0.02.

It is seen in Figure 16 that the states in frame space are accessed at about the same level, this is mainly because the binary scene pixel values are generated with equal probability (BIID). On the contrary, only 34 out of the 81 possible states from the feature space are accessed at all, which reflects the mutual exclusivities among the features for building up the state space and that some feature states do not exist (e.g. Fig 17). It is also interesting to note that though 34 states is a significant reduction from 512 states in characterizing the frame space, the performance is about the same. This implies the possibility of generalizing the input state space and reducing the required learning period. To test this possibility, the experiments were performed again with a much smaller number of trials (2,400 trials).

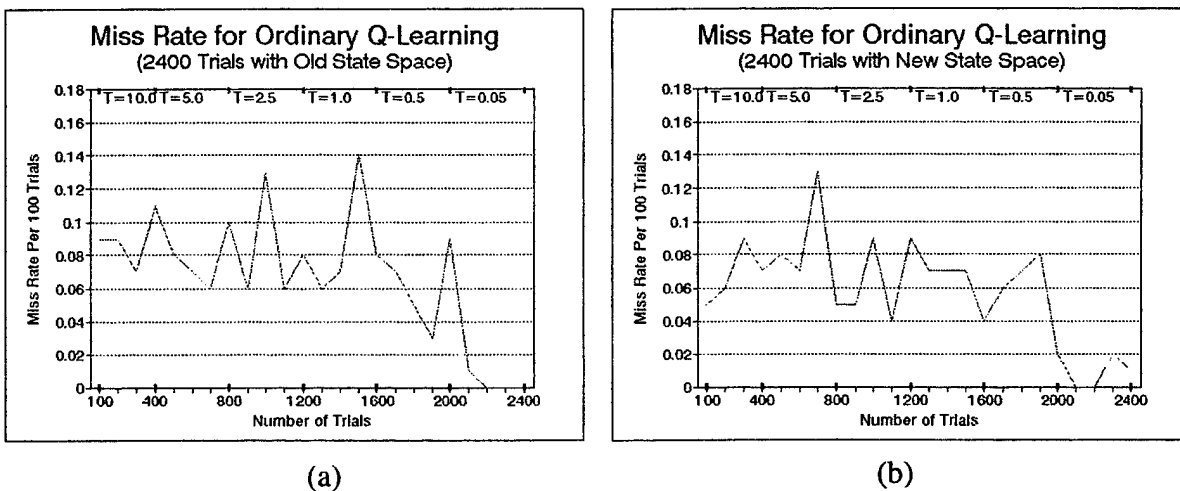


**Figure 17.** *Examples for conflicted features: states composed from these conflicted pairings are impossible to exist, thus are not accessed during the trials. The “?” stands for don’t care condition*

It is seen in Figures 18 and 19 that feature space performs well even with a small number of trials while frame space failed to give a satisfactory result due to insufficient exploration of its much larger number of state-action pairs. For comparison purpose, the experiments were performed again with a much larger number of trials (600,000). These computationally intensive version of the experiment resulted in a slightly improved performance with the miss rate reduced from range of 0-0.02 to 0-0.01 (Figures 20, 21). It is also shown from Figure 21 that feature space has a slightly poorer performance than that of the frame space, reflecting the fact that the generalized feature space failed to discriminate some of the distinct world states better interpreted in the more detailed frame space representation of perception-states.

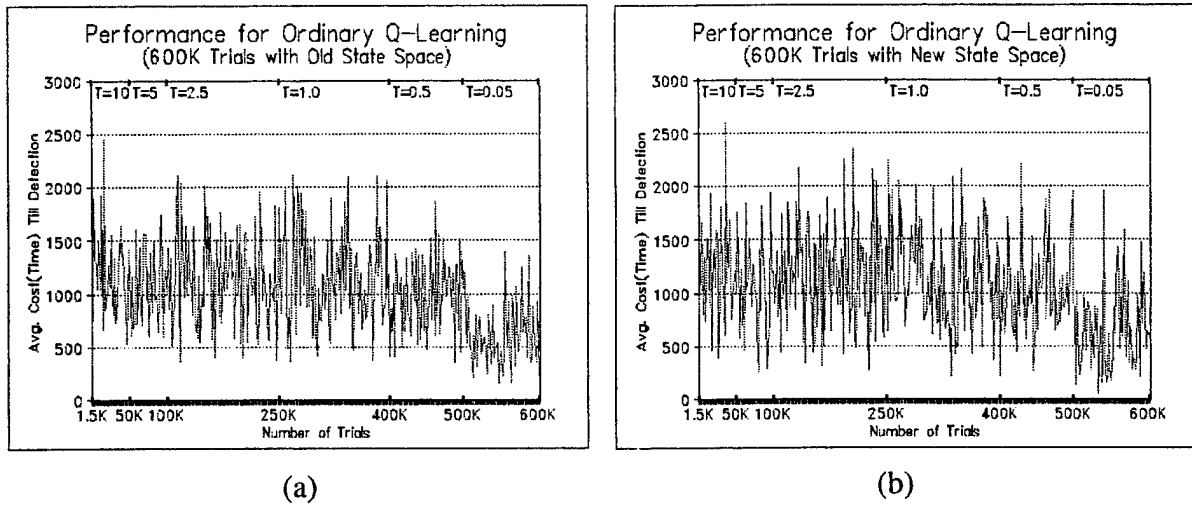


**Figure 18.** Performance using standard Q-learning with (a) frame space (b) feature space. Lack of a decreasing trend in frame space with  $T$  indicates inadequate exploration

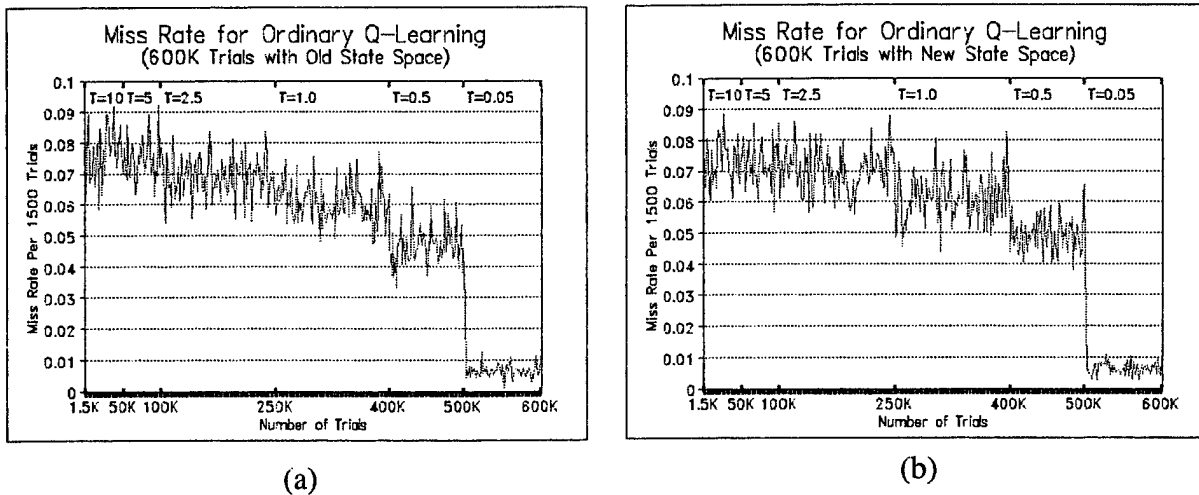


**Figure 19:** Miss rate by standard Q-Learning for 2400 trials with (a) frame space (b) feature space. Note good performance by both even in face of inadequate exploration in frame space case



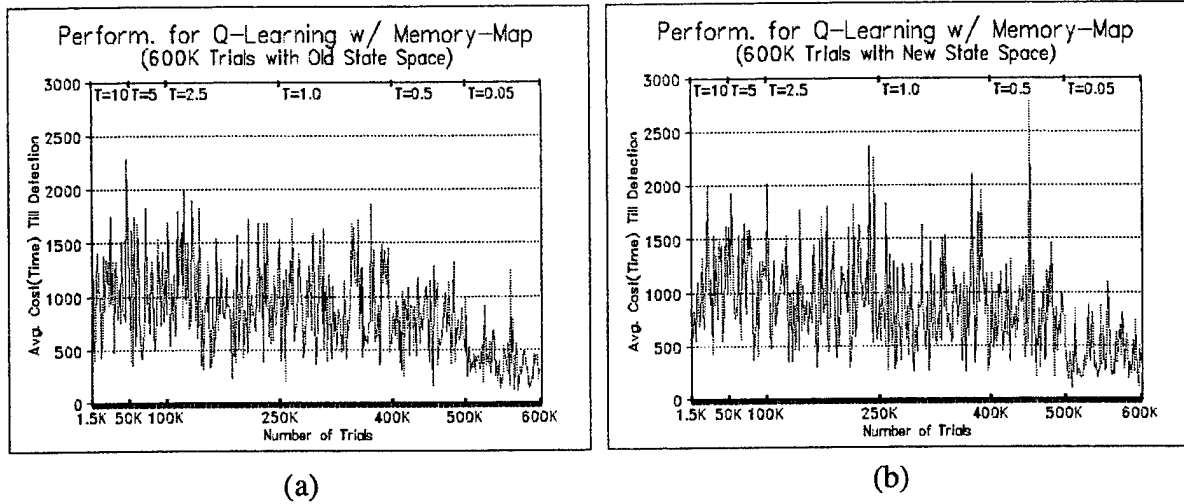


**Figure 20.** Performance (average of total cost till target is detected from 10 trials at a interval of 1500 trials) by ordinary Q-learning for 600K trials with (a) frame space (b) feature space

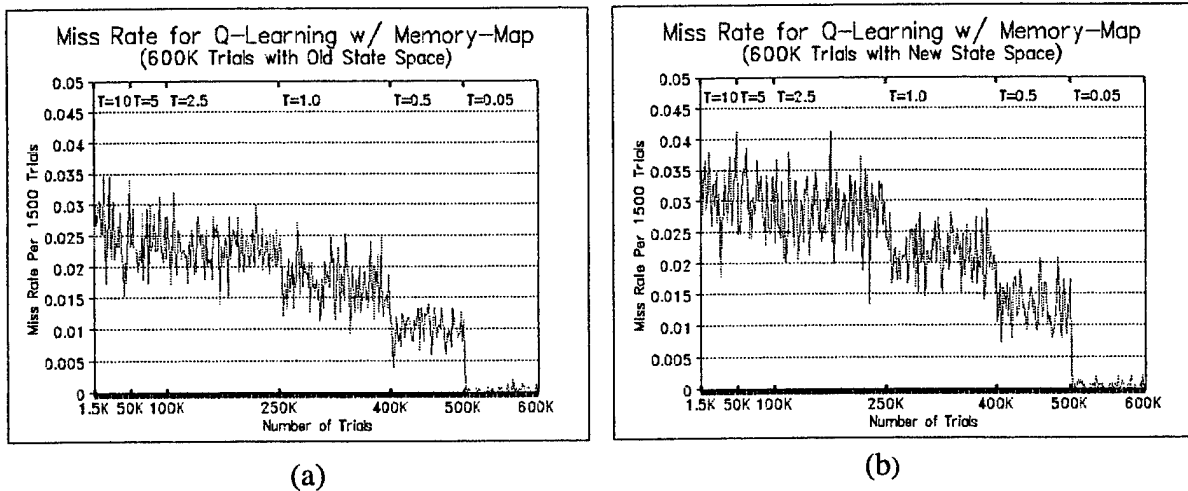


**Figure 21.** Miss rate (number of missed trials over the number of trials during the interval of 1500 trials) by ordinary Q-Learning for 600K trials with (a) frame space (b) feature space

We next turn to the fixation-window approach. This blocks the agent from inappropriate choices (revisiting scene locations that have been visited before and which contain no target). The extended version (600K) of the experiment is initially performed.

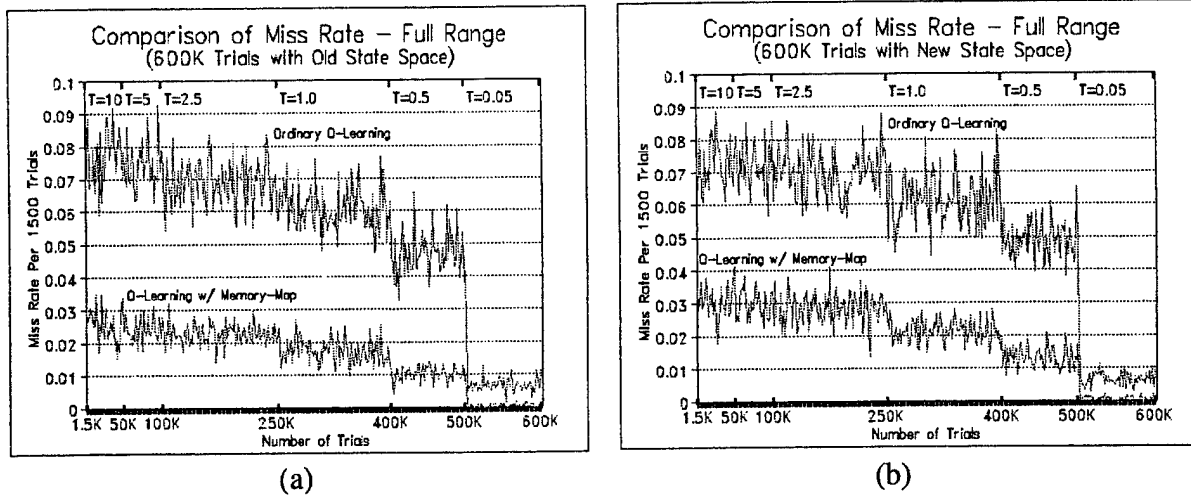


**Figure 22.** Performance (average of total cost till target is detected from 10 trials at a interval of 1500 trials) by Q-learning with fixation-window for 600K trials with (a) frame space (b) feature space

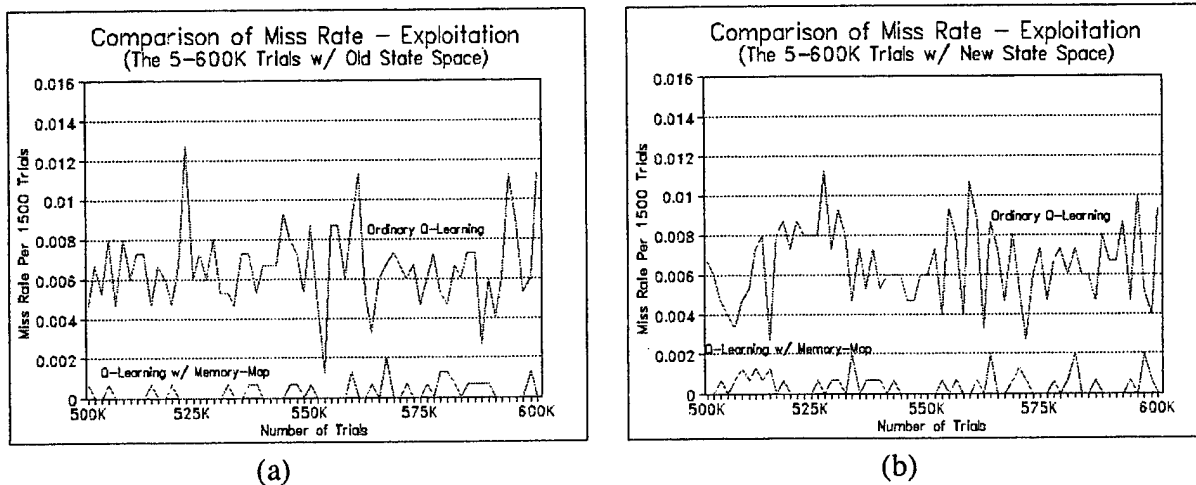


**Figure 23.** Miss rate using Q-Learning with fixation-window and (a) frame space (b) feature space

Comparing Figures 20 and 22, the improvement in performance is not significant. However, comparing the miss rate indexes shown in Figures 21 and 23, the fixation-window approach yields a greatly improved miss rate, even during the phase of random action selection ( $T=10$ ). Figure 24 demonstrates the overall improvement, and the comparison of the miss rate after the learning ( $T=0.05$ ) in Figure 25 show that the agent with the combination of Q-learning and fixation-window perception-state can detect targets with a much lower miss rate (0.002) than the one with standard Q-learning (0.01) in exploitation mode.



**Figure 24.** Comparison of the miss rate of 600K trials by standard Q-learning and 600K trials by Q-learning with fixation-window. Both (a) frame space and (b) feature space shown Q-learning with fixation-window has a lower miss rate than standard Q-learning



**Figure 25.** Comparison of the miss rate during exploration mode ( $T=0.05$ ) from the 600K trials by standard Q-Learning and Q-Learning with fixation-window. Both (a) frame space and (b) feature space shown. Q-Learning with fixation-window has a lower miss rate

The shorter version of the experiment (60,000 trials) is performed to test for consistency, and similar improvements from the fixation-window approach over the ordinary Q-learning are obtained as expected. The combination of Q-learning and memory-based approach is demonstrated to be an effective method for solving this non-Markovian POMDP control problem.

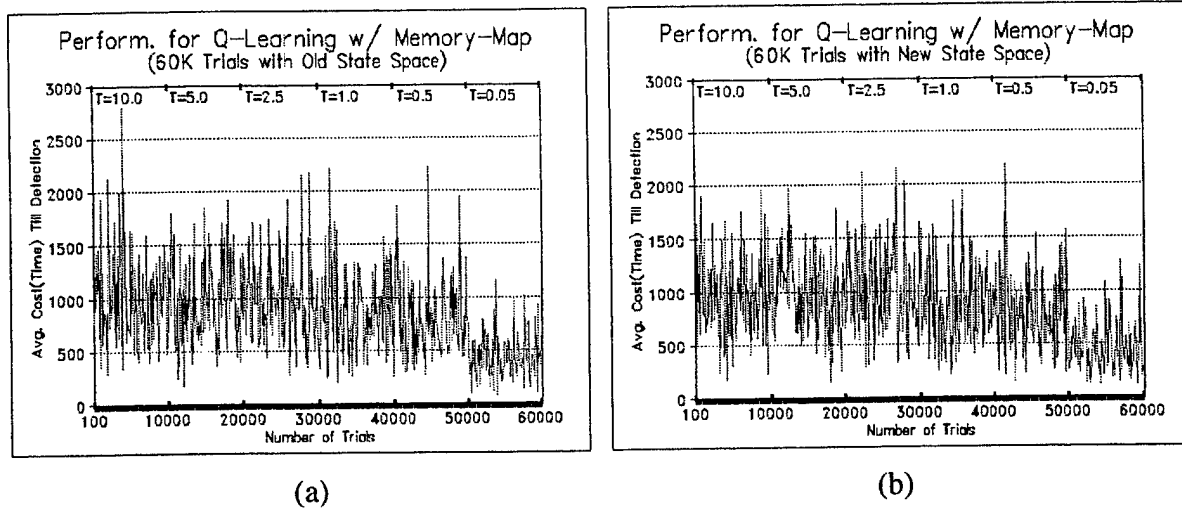


Figure 26. Performance by Q-Learning with fixation-window and (a) frame space (b) feature space

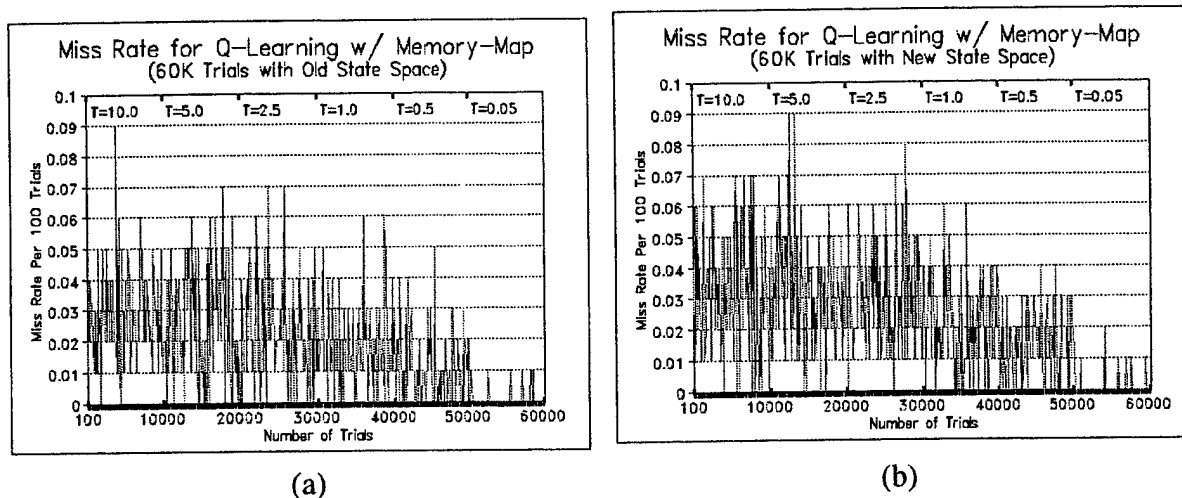


Figure 27. Miss rate by Q-Learning with fixation-window with (a) frame space (b) feature space

## 5.6 Discussion

This section develops algorithms and test results obtained in applying Q-learning strategies based on features derived from the current and past perceptions to various forms and scales of the gaze control problem in active vision. These experiments tested target detection performance in a rea-

sonably difficult simulation environment: small target (9 pixels), low SNR (-12.2 dB), no prior knowledge of the number or locations of targets present, and no structure to the background (white noise) to be exploited to eliminate regions. The state space was scaled to realistic dimensions for target detection applications.

The curse of dimensionality precluded perfect cumulative perceptual memory retention needed to permit Markov full-state updating as is required for guaranteed convergence using Q-Learning. Only the partially-observed or perception-state version of this learning regime is feasible even for the most constrained versions of the problem. Both past and current percept data were compressed to make for feasible state-action lookup table data structures.

The sequence of past observations was summarized in fixation-window memory, in which the locations of all fixations within a space-bounded sliding window moving with the fixation sequence are remembered. This feature was used to constrain the permissible actions, through the simple rule that having been determined not to be a target, a pixel location should not be revisited. Two partial representations of the current perception were employed as perception-states to reduce the number of states in the Q-function or evaluation function lookup table: frame memory and feature memory. The former stored the previous percept just as observed, the latter summarized the previous percept into a smaller number of perception-states indicating the degrees and compass orientations of partial matches with the target.

Table 3 below summarizes our results. The unconstrained random policy, with no helpful heuristics or learning at all, is to move at random, selecting the action from among all actions in the action set (first row of Table 3). Using the current percept as perception-state for policy selection feedback results in a policy fully taking the current percept into account, but without any memory of the results of past observations, is listed in the second row of Table 3 as Standard Q-Learning. As seen, this strategy cuts either distance or time costs roughly in half compared to the random-walk policy. Adding fixation-window memory, that is, memory of the previous fixation points within a sliding window, and using the feature space representation of the current percept leads to the results shown in the final row of Table 3. This strategy, designated as the fixation-window approach, is constrained in action space by the no-revisitation rule and in state space by the compression of the percept into partial-match features. Standard Q-learning is both more memory-intensive and has roughly 50% performance penalty (distance or time) relative to the fixation-window approach.

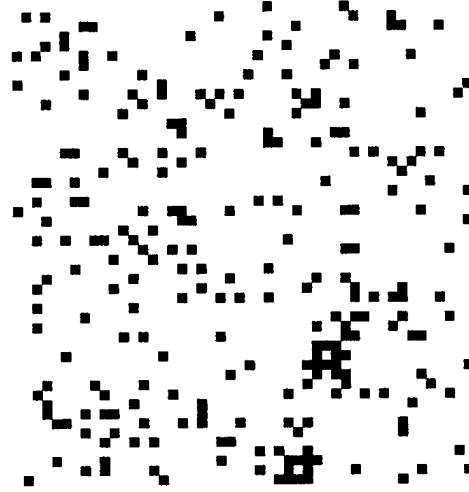
**Table 3: Experiment results with action set A and frame space**

Results of 100 Trials (After 60K Trials of Training)	Distance to target (pixels)	Time to target (fixations)
Unconstrained random	1236.77	616.90
Standard Q-learning	668.95	339.26
Fixation-window approach	412.08	206.56

The experiments reported thus far were performed in very noisy environments in order to challenge the learning schemes. To determine the effect of SNR on these trends, additional experiments were performed using higher SNR. A comparison field of view of a sample low SNR (-12.2 dB,  $P(\text{black}) = 1/2$ ) and higher SNR (-5.2 dB,  $P(\text{black})=1/10$ ) environments is shown below.



(a)  $P(\text{black}) = 0.5$



(b)  $P(\text{black}) = 0.1$

*50×50 sample environment with (a) -12.2 dB SNR, (b) -5.2 dB SNR*

Table 4 summarizes the results of experiments at these two SNR's and one intermediate value formed by setting  $P(\text{black}) = 0.7$  leading to -7.4 dB SNR. In each reinforcement learning case 600K training trials were employed and in all cases the last 300 trials were averaged. Three methods are compared: using the bare 3×3 fovea as the sensor and reinforcement learning based on fixation window feature (row 1), the modified wide field of view sensor (Figure 11) with reinforcement learning based on fixation window feature (row 2), and for a quasi-optimal non-adaptive reference, the modified wide field of view sensor with infinite memory and the greedy-immediate decision policy (row 3). Note that at the high SNR, the reinforcement learning algorithm performed almost as well as the quasi-optimal infinite-memory greedy-immediate reference strategy. This is encouraging, as realistic ATR or related recognition tasks usually occur at SNR's above 0 dB, a level not required for good performance here.

**Table 4: Experiment Results with different scene SNRs**

Method	-12.2 dB Time	-12.2 dB Distance	-7.4 dB Time	-7.4 dB Distance	-5.2 dB Time	-5.2 dB Distance
3×3 with RL	328.6	627.8	280.3	542.6	234.8	458.3
MWFOV with RL	291.5	560.8	192.3	373.1	149.1	288.5
MWFOV with GI	167.5	338.5	NA	NA	145.5	292.3

## Section 6: Error Residual Versus SARSA Learning

The previous sections demonstrate the feasibility and efficacy of reinforcement learning coupled with generalization produced by learned or fixed features as a gaze control paradigm for active perception. This section investigates several computational issues of reinforcement learning with function approximation within the context of behavior learning and selection. The objective is to examine the relative learning efficiency of these systems, which bears on significant importance to time-critical applications both commercial and military.

### 6.1 Reinforcement Learning with Function Approximation

Most reinforcement learning methods for autonomous robots require the identification of each world state and look-up table representations for policies and value functions in order for them to converge to optimality. However, look-up table representations are fundamentally unsuitable for learning in most practical problems. First, the state space of real world problems is often too large to allocate memory to the entire state space. As the dimension of the state space increases, the storage requirement becomes intractable. Second, look-up table representations typically learn the values of states one at a time, and the value of one particular state does not automatically influence the values of similar states. However, the state space of large-scale problems prohibits a learning system to visit all of the states, let alone update each one repeatedly to determine its correct value. Thus, in general lookup tables do not scale well for high dimensional problems (the *curse of dimensionality*).

There are a variety of methods that can represent value functions more efficiently and that are capable of valid generalization [Barnhill77,Franke82,Schumaker76]. One extreme case is when a closed-form expression can be found for the value function. However, the problems in which this can be performed are quite limited, such as when the world model is linear and the performance criterion is quadratic. Other methods settle for trying to represent the value function approximately. The basic principle of these approximate methods is to trade off optimality for efficiency, much like heuristic search algorithms do in artificial intelligence research. One approach of this type is to use a multigrid version of successive approximation (value iteration) that proceeds from coarse to fine grids. Another approach is to assume some underlying functional form for the value function and try to compute which actual function of this form best fits the Bellman equation as applied to actual data.

The fundamental difficulty with reinforcement learning with compact function approximation is that convergence to optimality, as in case of lookup tables, is no longer guaranteed. When combined with reinforcement learning, some such methods have been shown to be unstable in theory [Baird95,Gordon95] and in practice [BoyanMoore95]. On the other hand, other methods have been proven stable in theory [Sutton88,Peng94] and very effective in practice [Lin91,Tesauro92]. What are the requirements of a method in order to obtain good performance? In the following sections, we analyze two principal reinforcement learning algorithms namely: the residual gradient algorithm and the SARSA algorithm, both of which can be shown stable in theory [TsitsiklisRoy96]

and in practice [Sutton95].

## 6.2 Residual Algorithms and SARSA

For a prediction problem with a finite number of states, the optimal value function is the function that uniquely satisfies the Bellman equation:

$$V(x) = \langle r + \gamma V(y) \rangle \quad (\text{eq. 8})$$

where  $\langle \rangle$  denotes expectation and  $y$  is a successor state for a given state  $x$ . The *Bellman residual* (or *error*) for a value function  $V$  is defined to be the difference between the two sides of the Bellman equation (eq. 8). The *mean squared Bellman error* for a prediction problem with  $n$  states is thus:

$$E = (1/n) (\sum_x [\langle r + \gamma V(y) \rangle - V(x)]^2) \quad (\text{eq. 9})$$

The residual gradient algorithm [Baird95] attempts to minimize the Bellman error by performing gradient descent on the mean squared Bellman error,  $E$  (eq. 9).

For a deterministic prediction problem, the weight change  $\Delta W$ , after a transition from state  $x$  to state  $y$ , will be made according to:

$$\Delta W = -\alpha(r + \gamma V(y) - V(x))(\nabla_w [\gamma V(y) - V(x)]) \quad (\text{eq. 10})$$

where  $\alpha$  is the learning rate. It is clear that for a prediction problem with a finite number of states performing gradient descent on  $E$  guarantees convergence to a local minimum.

For a control problem with a finite number of states, the mean squared Bellman error will be:

$$E = (1/n) (\sum_x [\langle r + \gamma V(y) \rangle - Q(x,a)]) \quad (\text{eq. 11})$$

where  $V(x) = \max_a Q(x,a)$ . Similar to the deterministic prediction problem, weight change in a deterministic control problem after a transition from state  $x$  to state  $y$  resulting from taking action  $a$  will be made according to:

$$\Delta W = -\alpha(r + \gamma V(y) - Q(x,a))(\nabla_w [\gamma V(y) - Q(x,a)]) \quad (\text{eq. 12})$$

This will give rise to a similar algorithm performing gradient descent on  $E$  (eq. 11).

It is important to note that the derivation of (eq. 10) and (eq. 12) assumes that the dynamics of the underlying system is deterministic. If the dynamics were stochastic, the residual gradient algorithm would still converge, but not to a local minimum of the mean squared Bellman error. It will be something else, as was noted by Werbos [Werbos90]. Despite this, it has been suggested [Baird95] that the sheer convergence itself might still be a strong argument in favor of the algorithm as a viable candidate for solving stochastic prediction or control problems. We were curious to test this hypothesis.



For a stochastic problem, update rules (eq. 10) and (eq. 12) in general will have to be modified according to:

$$\Delta W = -\alpha(r + \gamma V(y) - V(x))(\nabla_w[\gamma V(y') - V(x)])$$

and

$$\Delta W = -\alpha(r + \gamma V(y) - Q(x, a))(\nabla_w[\gamma V(y') - Q(x, a)])$$

where  $y$  and  $y'$  are two successor states of  $x$ , each drawn independently from the distribution defined by the stochastic problem. This is required because an unbiased estimator of the product of two random variables can only be obtained by multiplying two independently generated unbiased estimators. To accomplish this a model of the problem must be known, either *a priori* or learned. This clearly increases the computational complexity of the algorithm. In addition, acquiring a model of the world may prove difficult, if not impossible, for many practical applications where state space is continuous. Furthermore, given that model learning is tractable, the algorithm might still be affected in undesired ways by errors in model acquisition, as shall be seen in the following sections.

### 6.3 The Random Walk Problem

The random walk problem, shown in Figure 28, is a simple prediction problem. There are seven states in which state D is the starting state. At each state a transition is made either to the left or to the right with equal probability except at states A and G, where the state transition is always to itself, i.e., states A and G are absorbing. The short-term reward is zero for every transition except the one from state F to state G when the reward 1 is delivered. The objective is to predict at each state the probability of reaching state G.

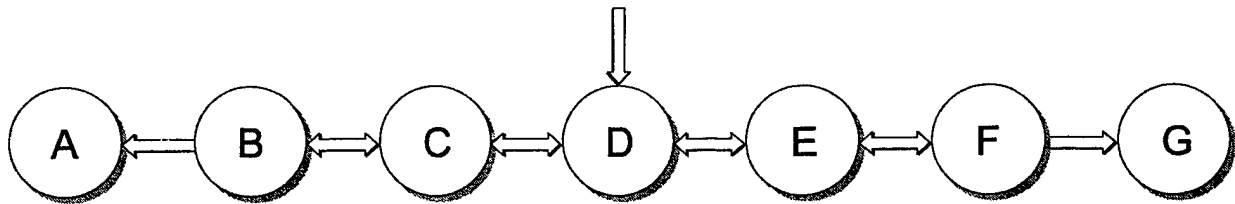


Figure 28. A random walk problem.

Previous studies [Sutton88, Sutton96] of the random walk problem have used lookup tables to represent the value function. In this experiment, the value function is represented by

$$V(x) = wx$$

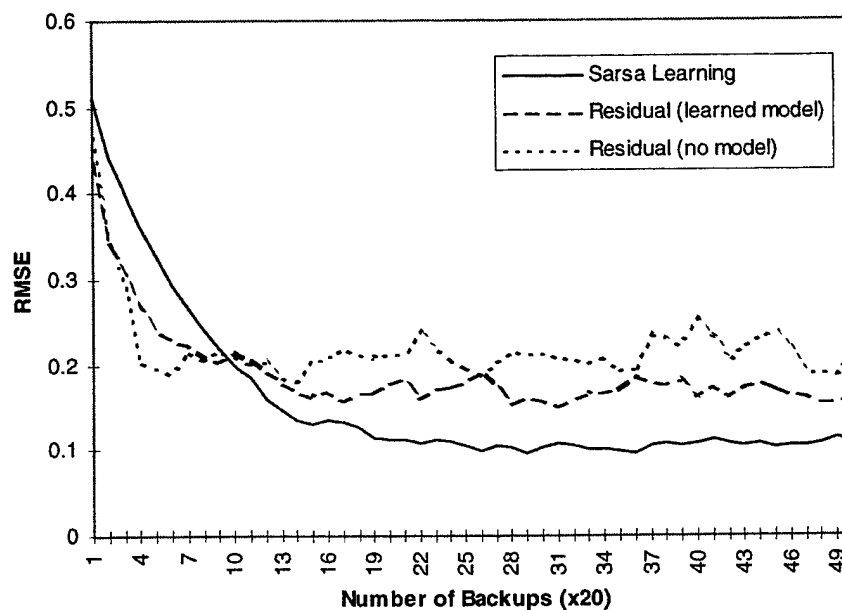
where  $w$  is a free parameter or weight, and  $x$  represents state. That is,  $V$  is a linear function of its input. States are encoded by integers from 0 to 7 corresponding to states from A to G, assuming values of states A and G are always zero. Note that the learning agent has no direct knowledge of what state it is in at each given moment. Instead, it only sees the encoding of the states. A trial be-

gins at state D and ends when either state A or state G is reached. The weight change is made after each transition.

For the random walk problem, the value of each state can be computed exactly. The ideal predictions for each of these states from B to F are: 1/6, 1/3, 1/2, 2/3, and 5/6. For a given approximation  $V$  to the ideal predictions, root mean-squared-error (RMSE) can be computed as a performance measure. The RMSE is computed anew after every 20 backups instead of a trial, where a backup is defined here to be a single weight update. The main reason is objectiveness. A close look at the residual gradient algorithm reveals that the update rule (eq. 10) can be rewritten as:

$$\Delta W = -\alpha(r + \gamma V(y) - V(x))(\nabla_w \gamma V(y) + \alpha(r + \gamma V(y) - V(x))\nabla_w(x))$$

which amounts to two backups in an update equation, as opposed one in SARSA learning. On the other hand, since there is only one weight in the approximation system, trace computation is negligible.



**Figure 29.** *Performance of random-walk by SARSA, Residual with a learned model, and Residual without a model.*

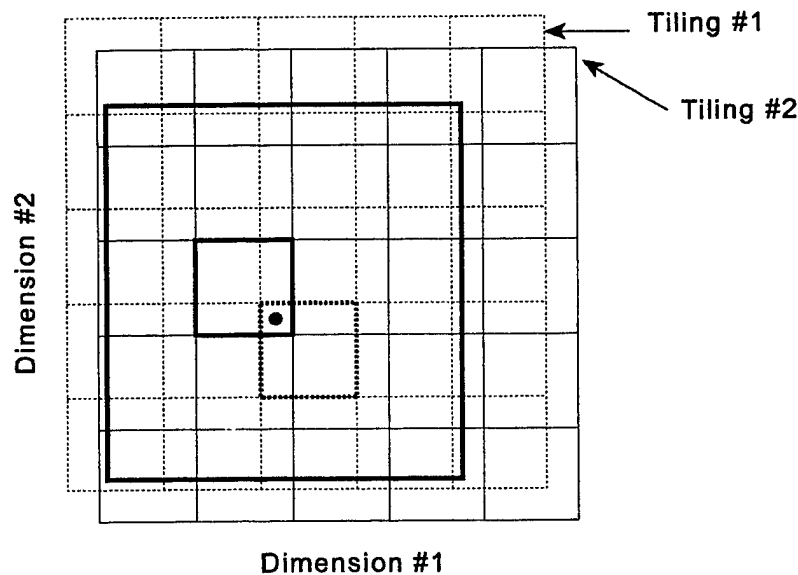
Figure 29 shows the performance of the random walk problem by SARSA, residual gradient with a learned model, and residual gradient without a model. The procedural (meta) parameters for these algorithms are:  $\lambda = 0.55$ ,  $\alpha = 0.002$  (SARSA);  $\alpha = 0.016$  (residual with a model);  $\alpha = 0.02$  (residual without a model). These curves are averaged over 50 runs. All the algorithms used the same initial random seeds. Figure 29 illustrates clearly the superior performance of the SARSA algorithm over the residual gradient algorithm. Both the SARSA and residual algorithms, with a learned model, continued to improve as learning proceeds. On the other hand, the residual gradient algorithm with-



The agent initially has no knowledge of the effect of its actions on what state it will occupy next, although it always knows its current state. Loosely stated, the objective is for the agent to discover a policy that enables it to obtain the maximum rate of reward received over time. More specifically, every time the agent occupies the goal region we may place it back in the starting region. In this case, the agent's objective is simply to discover a shortest path from the starting region to the goal region.

All of these methods apply more generally to a much wide range of learning control tasks, in which case the two dimensional continuous grid-world is simply a conceptual stand-in for the appropriate abstract state space for the actual problem and the compass direction moves used here represent the various control actions available.

The grid-world task has a continuous 2-D state space with an infinite number of states. To apply reinforcement learning some form of a function approximator is required. We used a set of four CMACs, one for each action. Unlike those neural networks such as multi-layer perceptrons which are global function approximators in that each hidden unit has a global receptive field, CMACs are sparse coarse coded function approximators. In particular, CMACs use a hashing algorithm to implement a distributed look-up table representation and a fixed amount of overlap of look-up table weights to reduce table size and generalization.



*Figure 31. CMACs involve multiple overlapping tilings of the state space.*

A CMAC uses multiple overlapping tilings of the state space to produce a feature representation for a final linear mapping where all the learning takes place (Figure 31). The overall effect is much like a RBF network with fixed radial basis functions, except that it is particularly efficient computationally. The significance of CMACs can be seen in recent work [TsitsiklisRoy96]. It is shown that for a fixed set of tilings (basis functions) from the CMACs, when coupled with  $TD(\lambda)$  returns,

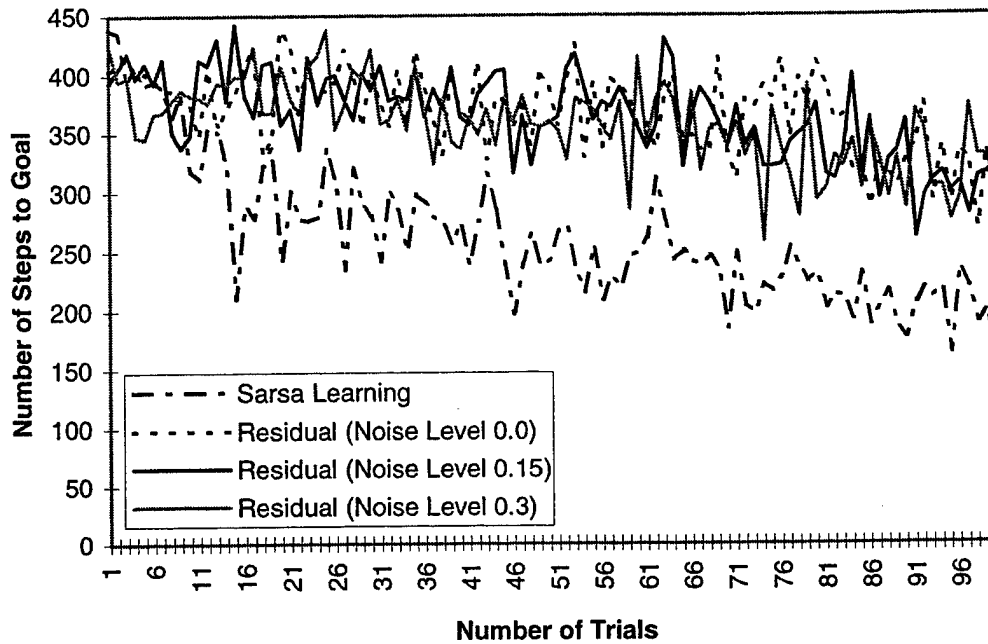
are guaranteed to converge to optimality in the LMS sense. Although this result applies only to prediction problems, it does shed light on control problems as well. It shows us that for any approximation scheme, convergence cannot be expected when updates do not follow actual sample trajectories.

In the grid-world problem we divided the two state variables each into ten evenly spaced intervals, thereby partitioning the state space into 100 regions or boxes. We added an eleventh row and column so that the tiling could be offset by a random fraction of an interval without leaving any state uncovered. This process was repeated five times, each with a different, randomly selected offset. The result was a total of  $11 \times 11 \times 5 = 605$  boxes. The state at any moment was represented by the five boxes, one per tiling, within which the state resided. One can think of the state representation as a feature vector with 605 features, exactly 5 of which are present at any point in time. The approximate action values are linear in this feature representation. Note that with the usual choice of binary basis functions, the CMAC approximate representation of the states results in a violation of the Markov property: many different nearby states produce exactly the same feature presentation.

It is important to note that the tilings need not be grids. For example, to dodge the “curse of dimensionality,” a possible technique is to ignore some dimensions in some tilings, i.e., to use hyperplanar slices instead of boxes. A second major technique is “hashing”, or varying resolution, a consistent random collapsing of a large set of tiles into a much smaller set. Through hashing, memory requirements are often reduced by large factors with little loss of performance. This is possible because high resolution is needed in only a small fraction of the state space, i.e., those where value functions are not smooth. Hashing liberates us from the curse of dimensionality in the sense that memory requirements need not grow exponentially with dimensionality, but need merely meet the real demands of the task.

An  $\epsilon$ -greedy policy was used to choose actions, where  $\epsilon$  was set to 0.15 in all the experiments. We used the heuristic to ensure sufficient exploration. The initial weights were set to lie between -0.1 and 0.1. A trial begins with the agent at a state in the starting region and ends when a state in the goal region has been reached. A starting state was uniformly, randomly generated from the starting region.

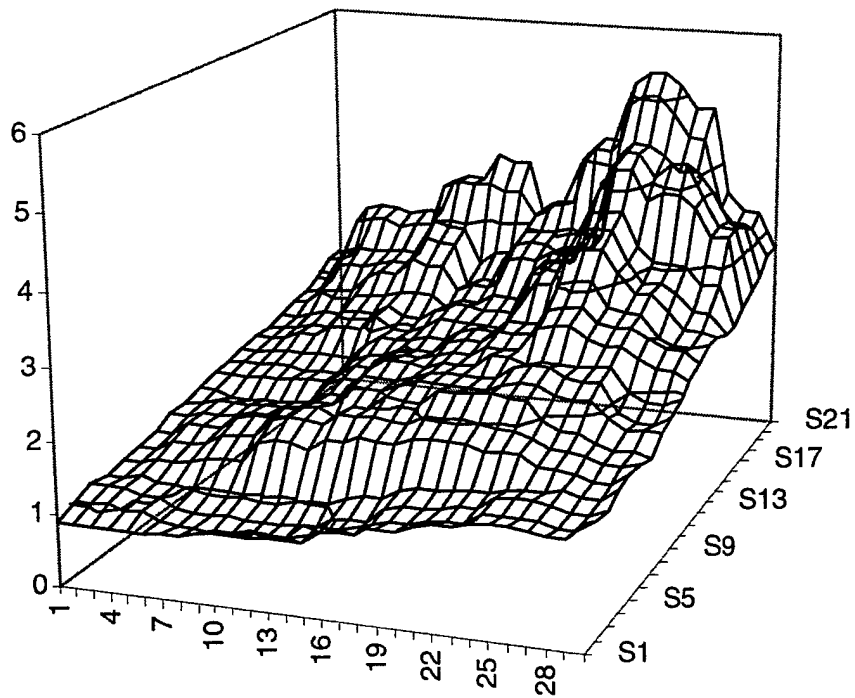
We applied SARSA and residual gradient algorithms to this task, each with procedural parameters selected after extensive search to give the best performance of each algorithm. Each algorithm was run for 100 trials. All algorithms used the same sets of random starting states and the same sets of initial random weights. The performance measure was the number of steps it takes for the agent to reach a goal state. Because of the stochastic transition nature, these steps were averaged over 10 traversals. That is, after each trial, the agent was placed at a starting state and then the greedy policy was followed until a goal state was reached or some maximum number of steps (500) was exceeded. This was done 30 times, and the average path length was recorded. This measure was then averaged over 10 runs to produce the results shown in Figure 32.



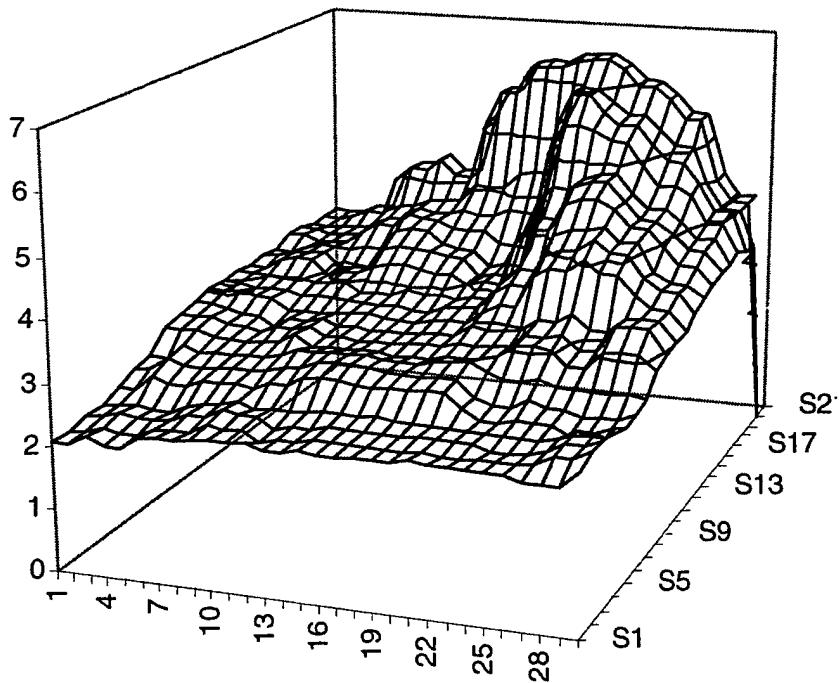
**Figure 32** *Performance of the continuous grid-world problem by SARSA and residual gradient algorithms.*

Several results are evident from Figure 32. First, the SARSA algorithm performed significantly better than the residual gradient algorithms. In particular, the results produced by the residual gradient algorithms assumed a known model available. If the residual gradient algorithms had to learn the model on-line, the results could have been worse. In addition, model learning would increase the overall computational complexity of these algorithms. To see how a model might affect performance, errors were added to the model. In general, errors may occur in two places: one in state transition estimation and the other in transition probability estimation. These errors always exist, particularly in high dimensional or continuous state spaces where compact approximation schemes must be used.

The performance of the residual gradient algorithms under erroneous model conditions is also shown in Figure 32. It can be seen that the performance degrades with increasing noise levels. Furthermore, the performance showed little or very slow improvement as more trials took place. This further verifies the results presented in the previous subsection. It casts serious doubt on applying residual gradient algorithms to solve stochastic Markov decision problems. In contrast, a model-free method such as SARSA would stand a much better chance to succeed in these problem domains.



**Figure 33** The value function computed by SARSA after 100 trials.



**Figure 34** The value function computed by the Residual Gradient algorithm using a perfect model after 100 trials.

Figure 33 plots the value function computed by SARSA after 100 trials. Similarly, Figure 34 plots the value function computed by the residual gradient algorithm using perfect model information. Clearly, SARSA created a better surface than the residual gradient method. The large valley area in the middle of the state space created by the residual gradient algorithm can cause the agent to move in a wrong direction, which certainly contributes to the relatively poor performance. In addition, the same value function shows flatness in the vicinity of the starting region, which is in sharp contrast with the value function computed by SARSA. Therefore, a “residual” agent can take a step in either direction instead of the preferred southward direction.

Much more empirical work is needed with these algorithms before a definitive conclusion can be drawn about their relative efficiency, particularly when function approximators are used. However, these experiments do provide strong evidence for two points: (1) the value function produced by SARSA is a better approximation to the optimal value function than that generated by the residual gradient algorithm, other things being equal, and (2) the residual gradient algorithms require a model in solving stochastic Markov decision problems, which causes these methods to be susceptible to unavoidable errors in model estimation.

## 6.5 Discussions

We have presented analytical and empirical studies evaluating the relative efficiency of SARSA and residual gradient algorithms using both prediction and stochastic control tasks. These studies provide valuable information for selecting reinforcement learning methods for machine vision gaze control in commercial settings.

These results showed consistent, significant, and sometimes large advantages of SARSA over residual gradient algorithms. They showed that to be useful in a stochastic Markov decision problem, the residual gradient algorithm must use model information, be it learned or known *a priori*. Furthermore, performance degrades with decreasing model accuracy. This bears on the question of the necessity of model information as control problems become non-deterministic. The main theoretical advantage of the residual gradient algorithm is that it converges in many cases to a minimum mean squared Bellman error solution, particularly when function approximators are used. SARSA, on the other hand, does not share this theoretical advantage. In practice, however, this may not be of great significance. All of our experimental results suggest far better performance is obtained with SARSA than with residual gradient algorithms.



## Section 7: Conclusions

Active vision has two basic components which are locked in a tight feedback loop: image sequence analysis and gaze control. Gaze control for active machine vision must support system integrity and performance over a wide range of operating conditions. This objective can be difficult to achieve due to several circumstances including the complexity of the performance objectives, the presence of dynamic uncertainty, and limited *a priori* model information. Under such conditions, it is very difficult or even impossible to design a control policy with fixed properties that meets the desired performance specifications. This report has addressed the problem of developing reinforcement learning algorithms with computational feasibility as well as effective generalizing capabilities for gaze control in active machine vision systems, and in the more demanding foveal systems.

The report began by giving a brief introduction to the field of reinforcement learning, a branch of machine learning, and various issues associated with reinforcement learning. It then described a particular theoretical framework within which to explore efficient computation and learning generalization. The description of dynamic programming provides a theoretical basis that serves both to explain the operation of a class of computational procedures, known as TD methods, and to relate them to existing theories of prediction, control, and learning. The report next presented a reinforcement learning method, namely SARSA, in conjunction with recurrent neural networks for gaze control. The role of the recurrent Elman neural networks is to provide a memory mechanism for learning salient history features. The efficacy of the method was empirically validated in a token active perception problem.

While an important demonstration of concept, reinforcement learning of history features simultaneous with learning appropriate valuation functions does not scale readily to realistic problems. Storage can be a problem, as the dimension of the set of potential learned features increases rapidly with problem scale and the size of the recurrent nets becomes excessive. For realistically scaled problems convergence time is also a potential problem, as the hierarchical search for good features and good valuation functions given the current, often inadequate, features cannot be expected to converge rapidly in the context of unguided trial and error training. For scaled problems the attractive but computationally burdensome optimal-seeking behavior of learned features may be replaced by heuristic fixed features. Often they are easily identified, and many orders of magnitude in storage and convergence time is saved. This approach was shown in a scaled problem to yield practical results, in which the learned behavior was shown to be quasi-optimal using small look-up tables and less than  $10^7$  trials.

Finally, SARSA and Residual Learning were compared in the context of neural generalization, and SARSA shown to be uniformly superior for the tests run. This is significant as residual learning has certain attractive asymptotic-optimality properties not shared by SARSA. As is often the case in search theory, asymptotic results are not necessary useful guides in finite searches.

In summary, we conclude that reinforcement learning coupled with fixed or learned features is a promisingly sound and practical paradigm for gaze control. Reward and punishment guidance, generalization, and full use of domain knowledge are all central to achieve practical, useful results for realistic problems.

## Section 8: References

- Albus, J.S., A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement, and Control*, September, 220-227, 1975.
- Asada, M., Noda, S., Taearatsumida, S., and Hosoda, K., (1995). Vision-Based Reinforcement Learning for Purposive Behavior Acquisition. *Proc. of IEEE Int. Conf. on Robotics and Automation*.
- Asada, M., Noda, S., Taearatsumida, S., and Hosoda, K., (1996a). Behavior Acquisition via Vision-Based Robot Learning. *Robotics Research, The Seventh International Symposium*. Springer. pp.1314-1319.
- Asada, M., Noda, S., Taearatsumida, S., and Hosoda, K., (1996b). Target Reaching Behavior Learning with Occlusion Detection and Avoidance for A Stereo Vision-Based Mobile Robot. *Proc. of ROBOLEARN96: An International Workshop on Learning for Autonomous Robots*
- Baird, L., Residual algorithms: Reinforcement learning with function approximation. *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.
- Bandera, C., (1990). *Foveal Machine Vision Systems*. Ph.D. dissertation, Department of Electrical and Computer Engineering, SUNY at Buffalo.
- Bandera, C., Scott, P., (1989). Foveal Machine Vision System. *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, Cambridge, MA, November.
- Bandera, C., and Scott, P., (1996) *Fusion of sensors that interact dynamically for exploratory development of robust, fast object detection and recognition - AFOSR STTR Phase I Progress Review*. Amherst Systems and University at Buffalo.
- Bandera, C., Residual Q-learning applied to visual attention, co-written with Dr. F.Vico, J. Bravo, Lt. M. Harmon, Capt. L. Baird. *Proceedings of the 13th International Conference on Machine Learning*, Bari, Italy, July 3, 1996.
- Barnhill, R.E., Representation and approximation of surfaces. *Mathematical Software III*, 69-120, 1977.
- Barto, A.G., Sutton, R.S., Watkins, C.J.C.H., *Learning and sequential decision making*. (COINS Technical Report 89-95). Department of Computer and Information Science, University of Massachusetts, Amherst, MA 1989.
- Barto, A.G, Bradke, S.J. and Singh, S.P., Learning to Act Using Real-Time Dynamic Programming, *Artificial Intelligence* 72, 81-138, 1995.
- Bellman, R., (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ.

Bellman, R.E., *Adaptive Control Processes*. Princeton University Press, 1961.

Bertsekas, D.P., A counter example to temporal-difference learning. *Neural Computation*, Vol. 7, 270-279, 1994.

Bertsekas, D.P., *Dynamic Programming: Deterministic and Stochastic Models*. Prentice Hall, Inc. 1987.

Bertsekas, D.P., Tsitsiklis, J.N., *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, Inc. 1989.

Boyan, J., Moore, A., Generalization in reinforcement learning: Safely approximating the value function. *NIPS-7*, 1995.

Boyan, J., Moore, A., Generalization in reinforcement learning: Safely approximating the value function. *NIPS-7*, 1995.

Boyan, J., Moore, A.W., Learning evaluation functions for large acyclic domains. *Proceedings of the Thirteen International Conference on Machine Learning*, 1996.

Carpenter, G.A., and Grossberg, S., (1987a) A massively parallel architecture for a self-organizing neural pattern recognition machine. *Comput. Vision, Graphics Image Process.*, Vol 37, pp54-115

Carpenter, G.A., and Grossberg, S., (1987b) ART2 : self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, Vol 26, pp 4919-4930.

Fagg, A.h., Lotspeich, D., and Bekey, G.A., (1994). A Reinforcement-Learning Approach to Reactive Control Policy Design for Autonomous Robots. *1994 IEEE Conference on Robotics and Automation*.

Franke, R., Scattered data interpolation: Tests of some methods. *Mathematics of Computation* 38(157), 1982.

Fu, K.S., Learning Control Systems – Review and Outlook. *IEEE Transactions on Automatic Control*, 210-221, April 1970.

Girosi, F., Jones, M., Poggio, T., Regularization theory and neural networks architectures. *Neural Computation* 7, 219-269, 1995.

Gordon, G., Stable function approximation in dynamic programming. *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

Kaelbling, L.P., *Learning in embedded systems*. Ph.D. Dissertation, Dept. of Computer Science, Stanford University, CA 1990.

- Kaelbling, L.O., Littman, M.L., and Moore, A.W., (1996). Reinforcement Learning : A Survey. *Journal of Artificial Intelligence Research* 4. 237-285.
- LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., and Jackel, L.D., Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation* (1) 541-551, 1989.
- Lin, L.J., Self-improving reactive agents based on reinforcement learning, planning, and teaching. *Machine Learning*, 8(3/4), 293-321, 1992.
- Lin L-J., (1991) Programming robots using reinforcement learning and teaching. *Proceedings of AAAI-91* pp. 781-786.
- Miller, W. T. III, Glanz, F. H., Kraft, L. G. III, Application of a general learning algorithm to the control of robotic manipulators. *The International Journal of Robotics Research* 6(2), 84-98, 1987.
- Minsky, M.L., Steps toward artificial intelligence. *Proceedings IRE* 49, 8-30, 1961.
- Mitchell, T., Keller, R., Kedar-Cabelli, S., Explanation-based learning: A unifying view. *Machine Learning*, 1, 47-80, 1986.
- Moore, A.W., (1994). The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. In Cowan, J.D., Tesauro, G., & Alspector, J. (eds), *Advances in Neural Information processing Systems 6*, pp.711-718 San Mateo, CA. Morgan Kaufman.
- Moore, A.W., Atkeson, C.G., Memory-based reinforcement learning: Converging with less data and less real time. *Proceedings of NIPS\*92*, 1992.
- Narendra, K.S., Thathachar, M.A.L., *Learning Automata: An Introduction*. Englewood Cliffs, NJ: Prentice Hall 1989.
- Pandya, A.D., Macy, Robert B., (1996) *Pattern Recognition with Neural Networks in C++*, IEEE Press.
- Peng, J., *Efficient dynamic programming based learning for control*. Ph.D. Dissertation, Northeastern University, Boston, MA 1994.
- Peng, J., Williams, R.J., Incremental multi-step Q learning. *Machine Learning*, 22, 283-290, 1996.
- Peng, J., Williams, R.J., Incremental multi-step Q learning. *Proceedings of the Eleventh International Conference on Machine Learning*, 226-232, 1994.
- Peng, J., Williams, R.J., Efficient learning and planning within the Dyna framework. *Adaptive Behavior Vol. I No. 4*, 1993.

Powell, M.J.D., Radial basis functions for multivariable interpolation: A review. In *Algorithms for Approximation*. (Mason, C. & Cox, M.G., Eds) Clarendon, Oxford 143-167, 1987.

Ross, S., *Introduction to Stochastic Dynamic Programming*. Academic Press, New York, 1983.

Schumaker, L.L., Fitting surfaces to scattered data. *Approximation Theory II*. Academic Press 203-268, 1976.

Rummery, G., Niranjan, M., On-line Q learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University, 1994.

Schwartz, A., A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the Tenth International Conference on machine Learning*, 298-305, 1993.

Shibata, K., Nishimo, T., and Okabe, Y., (1995). Active Perception Based on Reinforcement Learning. *Proceedings of WCNN'95*. Washington D.C., vol.2, pp.170-173.

Sutton, R.S., Personal communication, NSF Reinforcement Learning Workshop, Harpers Ferry, WV, April 12-14, 1996.

Sutton, R.S., Generalization in reinforcement learning: successful examples using sparse coarse coding. *Proceedings of NIPS*, 1995.

Sutton, R.S., Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning*, 216-224, 1990.

Sutton, R.S., Learning to predict by the methods of temporal differences. *Machine Learning*, 9-44, 1988.

Tsitsiklis, J.N., Van Roy, B., *An analysis of temporal-difference learning with function approximation*. LIDS-P-2322, Laboratory for Information and Decision Systems, MIT, March 1996.

Tesauro, G., Practical issues in temporal difference learning. In: J.E.Moody, S.J.Hanson, R.P.Lippmann (Eds.), *Advances in Neural Information Processing Systems 4*, 259-266, 1992.

Watkins, C.J.C.H., *Learning from delayed rewards*. Ph.D. Dissertation, King's College, UK 1989.

Watkins, C.J.C.H., & Dayan, P., (1992) Q-Learning. *Machine Learning*, 8(3), 279-292.

Werbos, P.J., Consistency of HDP applied to a simple reinforcement learning problem. *Neural Networks* (3) 179-189, 1990.

Whitehead, S.D., and Ballard, D.H., (1991) Learning to perceive and act by trial and error. *Machine Learning* 7, 45-83.

Williams, R. J., Peng, J., Function optimization using connectionist reinforcement learning algorithms. *Connection Science* 3(3), 1991.

Zurada, J.M., (1992) *Introduction to Artificial Neural Systems*. West Publishing Company.

<b>REPORT DOCUMENTATION PAGE</b>			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE  31, May1997		3. REPORT TYPE AND DATES COVERED  Final Technical Report
4. TITLE AND SUBTITLE  A Reinforcement Learning Approach to Control			5. FUNDING NUMBERS  Contract No.: N00014-96-C-0321	
6. AUTHOR(S)  Cesar Bandera, Peter Scott, Jing Peng				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)  Amherst Systems Inc. 30 Wilson Road Buffalo, NY 14221			8. PERFORMING ORGANIZATION REPORT NUMBER  625-9160004, rev a	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)  Office of Naval Research ONR 251 WDW, Ballston Tower One 800 No. Quincy Street Arlington, VA 22217-5660			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. Supplemental Notes				
12a. DISTRIBUTION / AVAILABILITY STATEMENT			12a. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) Active perception strategies are necessary for goal-driven allocation of available resources to improve relevant information acquisition and optimize overall system performance. In addition to being both goal and data driven, these strategies must also account for the fact that information acquisition is inherently a partially observable Markov decision problem. This report describes an efficient, scalable reinforcement learning approach to the control of autonomous active vision that also satisfies the more stringent requirements of foveal machine vision. Foveal vision offers images with both wide field of view, useful for rapid detection, and a high acuity zone, useful for accurate recognition, without the overhead and errors inherent in dynamic registration of data from multiple sensors. However, space-variant data acquisition inherent with foveal retinotopologies necessitates deployment of refined intelligent gaze control techniques. This report first lays a theoretical foundation for reinforcement learning. It then introduces the SARSA algorithm in conjunction with history augmentation as an effective learning control method for visual attention. The system is shown to perform well in both high and low SNR ATR environments. Reinforcement learning coupled with history features appears to be both a sound foundation and a practical scalable base for gaze control.				
14. SUBJECT TERMS Reinforcement Learning, Robotics, Active Vision, Hierarchical Processing, Foveal Vision, Multiresolution			15. NUMBER OF PAGES  60	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT  Unlimited	